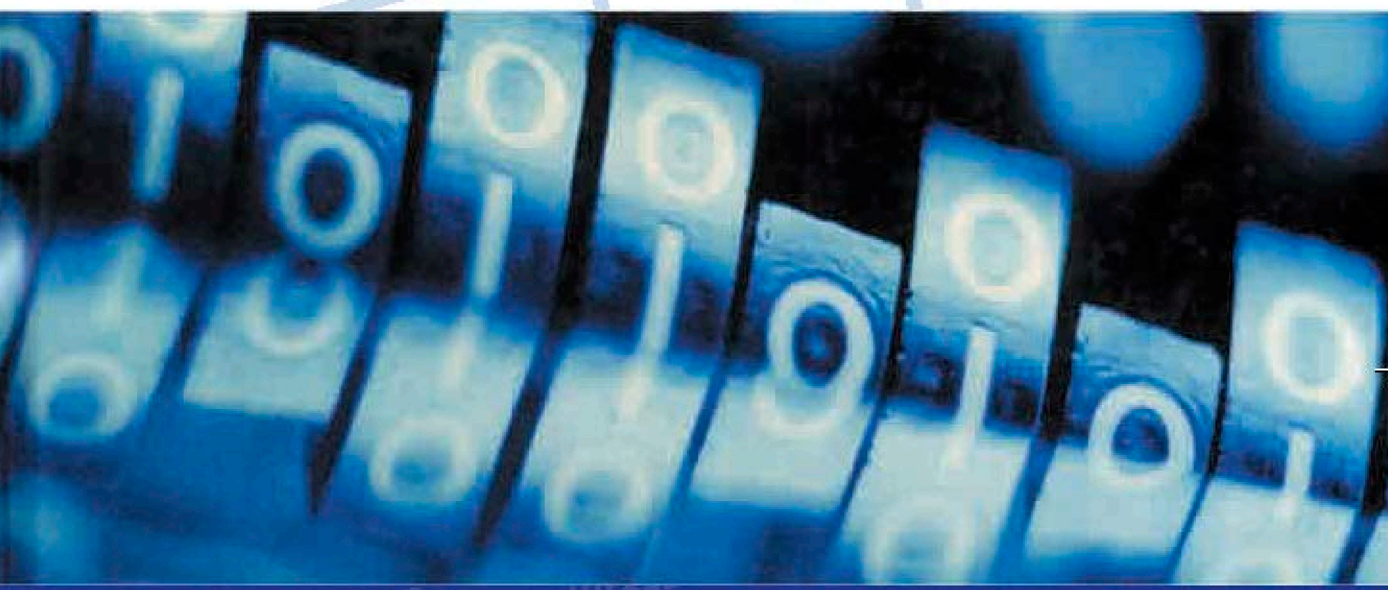
DISEÑO DIGITAL

TERCERA EDICIÓN



module
input A,B,select;
output OUT;
bufif0 (OUT,B,select;
endmodule

M. MORRIS MANO

INCLUYE CD-ROM

DISEÑO DIGITAL

TERCERA EDICIÓN

M. Morris Mano
CALIFORNIA STATE UNIVERSITY, LOS ANGELES

TRADUCCIÓN
Roberto Escalona García
Ingeniero Químico
Universidad Nacional Autónoma de México

REVISIÓN TÉCNICA
Gonzalo Duchén Sánchez
Sección de Estudios de Postgrado e Investigación
Escuela Superior de Ingeniería Mecánica y Eléctrica
Unidad Culhuacán
Instituto Politécnico Nacional

México • Argentina • Brasil • Colombia • Costa Rica • Chile • Ecuador
España • Guatemala • Panamá • Perú • Puerto Rico • Uruguay • Venezuela

Datos de catalogación bibliográfica

MORRIS, MANO, M.

Diseño digital. Tercera edición

PEARSON EDUCACIÓN, México, 2003

ISBN: 970-26-0438-9
Área: Universitarios

Formato: 18.5 × 23.5 cm Páginas: 536

ISBN 0-13-062121-8

Edición en español
Editor: Guillermo Trujano Mendoza
e-mail: guillermo.trujano@pearsoned.com
Editor de desarrollo: Felipe Hernández Carrasco
Supervisor de producción: José D. Hernández Garduño

Edición en inglés
Vice President and Editorial Director, ECS: Marcia J. Horton
Publisher: Tom Robbins
Acquisitions Editor: Eric Frank
Editorial Assistant: Jessica Romeo
Vice President and Director of Production and Manufacturing, ESM: David W. Riccardi
Executive Managing Editor: Vince O'Brien
Managing Editor: David A. George
Production Editor: Lakshmi Balasubramanian
Director of Creative Services: Paul Belfanti
Creative Director: Carole Anson
Art Director and Cover Designer: Jonathan Boylan
Art Editor: Adam Velthaus
Manufacturing Manager: Trudy Pisciotti
Manufacturing Buyer: Lisa McDowell
Marketing Manager: Holly Stark
Marketing Assistant: Karen Moon

TERCERA EDICIÓN, 2003

Atlacomulco No. 500, 5° piso
Col. Industrial Atoto
53519, Naucalpan de Juárez, Edo. de México
E-mail: editorial.universidades@pearsoned.com

Cámara Nacional de la Industria Editorial Mexicana. Reg. Núm. 1031

ISBN 970-26-0438-9

Impreso en México. *Printed in Mexico.*

1 2 3 4 5 6 7 8 9 0 - 06 05 04 03

CONTENIDO

PREFACIO

El diseño digital se ocupa del diseño de circuitos electrónicos digitales. Los circuitos digitales se emplean en el diseño y construcción de sistemas como computadoras digitales, comunicación de datos, grabación digital y muchas otras aplicaciones que requieren hardware digital. Este libro presenta las herramientas básicas para el diseño de circuitos digitales y proporciona los conceptos fundamentales empleados en el diseño de sistemas digitales. Será muy útil como texto de un curso introductorio dentro de un programa de estudios de ingeniería eléctrica, ingeniería en computación o ciencias de la computación.

Muchas de las características de esta tercera edición siguen siendo las mismas que las de ediciones anteriores, salvo por el reacomodo del material o cambios en el enfoque debidos a avances en la tecnología. Los circuitos combinacionales se abordan en un capítulo en vez de dos, como en la edición anterior. El capítulo sobre circuitos secuenciales hace hincapié en el diseño con flip-flops *D* en lugar de flip-flops *JK* y *SR*. El material sobre memoria y lógica programable se ha combinado en un capítulo. El capítulo 8 se ha modificado para incluir los procedimientos de diseño en el nivel de transferencia de registros (RTL).

La principal modificación en la tercera edición es la inclusión de secciones sobre el Lenguaje de Descripción de Hardware (HDL) Verilog. El material sobre HDL se ha insertado en secciones aparte de modo que el profesor pueda decidir cómo incorporarlo a su curso. La presentación tiene un nivel apropiado para estudiantes que están aprendiendo circuitos digitales y un lenguaje de descripción de hardware al mismo tiempo.

- Los circuitos digitales se introducen en los capítulos 1 al 3; en la sección 3-9 se hace una introducción a Verilog HDL.
- El HDL se trata más a fondo en la sección 4-11 después de estudiarse los circuitos combinacionales.
- Los circuitos secuenciales se tratan en los capítulos 5 y 6, con ejemplos en HDL en las secciones 5-5 y 5-6.
- La descripción HDL de la memoria se presenta en la sección 7-2.

- Los símbolos RTL empleados en Verilog HDL se presentan en la sección 8-2.
- Se dan ejemplos de descripciones HDL en los niveles RTL y estructural en las secciones 8-5 y 8-8.
- La sección 10-10 cubre el modelado en el nivel de interruptores que corresponde a los circuitos CMOS.
- La sección 11-9 complementa los experimentos de hardware del capítulo 11 con experimentos en HDL. Ahora los circuitos diseñados en el laboratorio se pueden verificar con la ayuda de componentes de hardware o con simulación HDL, o ambas cosas.

El CD-ROM que acompaña al libro contiene los archivos en código fuente Verilog HDL para los ejemplos del libro, además de dos simuladores cortesía de SynaptiCAD. El primer simulador es VeriLogger Pro, un simulador tradicional de Verilog que servirá para simular los ejemplos en HDL del libro y comprobar las soluciones a los problemas en HDL. El segundo es un nuevo tipo de tecnología de simulación: un Simulador Interactivo. Este programa permite a los ingenieros simular y analizar ideas de diseño antes de contar con un modelo o esquema completo de simulación. Esta tecnología es de especial utilidad para los estudiantes, porque pueden introducir rápidamente ecuaciones booleanas y de entradas de flip-flop *D* o latch para verificar la equivalencia o experimentar con flip-flops y diseños de latches. Se incluyen cursos rápidos en forma de archivos HTML en la presentación Flash del CD-ROM, y en forma de archivos MS Word en el directorio instalado SynaptiCAD bajo Book Tutorials.

Hay más recursos en un sitio Web acompañante, http//www.prenhall.com/mano. En él se pueden bajar todos los ejemplos en Verilog HDL del libro, todas las figuras y tablas del libro en formato PDF, cursos breves sobre el uso del software Verilog del CD-ROM, y mucho más.

A continuación se describen brevemente los temas que se tratan en cada capítulo, haciendo hincapié en los cambios efectuados para la tercera edición.

El **capítulo 1** presenta los diversos sistemas binarios apropiados para representar información en sistemas digitales. Se explica el sistema numérico binario y se ilustran los códigos binarios. Se dan ejemplos de suma y resta de números binarios con signo y de números decimales en BCD.

El **capítulo 2** introduce los postulados básicos del álgebra booleana y muestra la correlación entre las expresiones booleanas y los diagramas lógicos correspondientes. Se investigan todas las posibles operaciones lógicas con dos variables para encontrar las compuertas lógicas más útiles en el diseño de sistemas digitales. En este capítulo se mencionan las características de las compuertas de circuitos integrados pero se deja para el capítulo 10 un análisis más a fondo de los circuitos electrónicos de dichas compuertas.

El **capítulo 3** cubre el método del mapa para simplificar expresiones booleanas. Ese método también sirve para simplificar circuitos digitales construidos con compuertas AND-OR, NAND o NOR. Se consideran todos los demás circuitos que pueden formarse con dos niveles de compuertas y se explica su método de implementación. Se introduce Verilog HDL junto con ejemplos sencillos de modelado en el nivel de compuertas.

El **capítulo 4** bosqueja los procedimientos formales para analizar y diseñar circuitos combinacionales. Se presentan como ejemplos de diseño algunos componentes básicos empleados en el diseño de sistemas digitales, como sumadores y convertidores de código. Se explican las funciones de lógica digital de uso más común, como el sumador y restador paralelo, los decodificadores, codificadores y multiplexores, con ilustraciones de su uso en el diseño de circuitos combinacionales. Se dan ejemplos en HDL de modelado en el nivel de compuertas, flujo de datos y comportamiento, para mostrar las diferentes formas con que se cuenta para descri-

bir circuitos combinacionales en Verilog HDL. Se presenta el procedimiento para escribir un conjunto sencillo de pruebas que suministre estímulos a un diseño HDL.

El **capítulo 5** delinea los procedimientos formales para el análisis y diseño de circuitos secuenciales sincrónicos con reloj. Se presenta la estructura de compuertas de varios tipos de flip-flops, junto con una explicación de las diferencias entre el disparo por nivel y por borde. Se incluyen ejemplos específicos para explicar cómo se deduce la tabla de estados y el diagrama de estados al analizar un circuito secuencial. Se presentan varios ejemplos específicos haciendo hincapié en los circuitos secuenciales que utilizan flip-flops tipo *D*. Se explica el modelado del comportamiento de circuitos secuenciales en Verilog HDL, proporcionando ejemplos en HDL que ilustran los modelos Mealy y Moore de circuitos secuenciales.

El **capítulo 6** se ocupa de diversos componentes de los circuitos secuenciales, como registros, registros de desplazamiento y contadores. Estos componentes digitales son los bloques básicos con los que se construyen sistemas digitales más complejos. Se presentan descripciones de registros de desplazamiento y contadores en HDL.

El **capítulo 7** trata la memoria de acceso aleatorio (RAM) y los dispositivos lógicos programables. Se explica la decodificación de memoria y los esquemas de corrección de errores. Se presentan dispositivos programables combinacionales y secuenciales, como ROM, PAL, CPLD y FPGA.

El **capítulo 8** se ocupa de la representación de sistemas digitales en el nivel de transferencia de registros (RTL). Se introduce el diagrama de máquina de estados algorítmica (ASM). Varios ejemplos ilustran el uso del diagrama ASM, la representación RTL y la descripción HDL en el diseño de sistemas digitales. Este capítulo es el más importante del libro, pues prepara al estudiante para proyectos de diseño más avanzados.

El **capítulo 9** presenta procedimientos formales para el análisis y diseño de circuitos secuenciales asincrónicos. Se bosquejan métodos para implementar un circuito secuencial asincrónico como circuito combinacional con retroalimentación. También se describe una implementación alterna que usa latches SR como elementos de almacenamiento en circuitos secuenciales asincrónicos.

El **capítulo 10** presenta las familias de lógica digital más comunes para circuitos integrados. Se analizan los circuitos electrónicos de la compuerta común en cada familia, empleando teoría de circuitos eléctricos. Se requieren conocimientos básicos de electrónica para entender cabalmente el material de este capítulo. Los ejemplos de descripciones Verilog en el nivel de interruptores ilustran la capacidad para simular circuitos construidos con transistores MOS y CMOS.

El **capítulo 11** delinea experimentos que se pueden efectuar en el laboratorio con hardware que se consigue con facilidad en el comercio. Se explica el funcionamiento de los circuitos integrados empleados en los experimentos haciendo referencia a diagramas de componentes similares presentados en capítulos anteriores. Cada experimento se presenta de manera informal y se espera que el estudiante dibuje el diagrama de circuitos y formule un procedimiento para verificar el funcionamiento del circuito en el laboratorio. La última sección complementa los experimentos con los correspondientes en HDL. En vez de construir los circuitos en hardware, o además de hacerlo, el estudiante podrá utilizar el software de Verilog HDL incluido en el CD-ROM para simular y verificar el diseño.

El **capítulo 12** presenta los símbolos gráficos estándar que una norma ANSI/IEEE recomienda para las funciones lógicas. Estos símbolos gráficos se crearon para componentes SSI y MSI, con el fin de que el usuario tenga posibilidad de reconocer cada función a partir del símbolo gráfico único que tiene asignado. Este capítulo muestra los símbolos gráficos estándar de

los circuitos integrados que se emplean en los experimentos de laboratorio. Los diversos componentes digitales que se representan en todo el libro son similares a los circuitos integrados comerciales. Sin embargo, el texto no menciona circuitos integrados específicos, con la excepción de los capítulos 11 y 12. La aplicación práctica del diseño digital será más provechosa si se efectúan los experimentos sugeridos en el capítulo 11 mientras se estudia la teoría presentada en el texto.

Cada capítulo incluye una lista de referencias y un conjunto de problemas. Al final del libro se brindan las respuestas de problemas seleccionados como ayuda para el estudiante y para el lector independiente. El profesor puede solicitar un manual de soluciones (en inglés) a la casa editorial.

Quiero agradecer a Charles Kime por darme a conocer Verilog. Mi mayor deuda es con Jack Levine por guiarme y revisar las secciones, ejemplos y soluciones a problemas de todo el material sobre Verilog HDL. Gracias a Tom Robbins por su estímulo para decidirme a escribir la tercera edición, y a mi editor, Eric Frank, por su paciencia durante toda la revisión. Quiero expresar mi aprecio a Gary Covington y Donna Mitchell, quienes proporcionaron el CD-ROM de SynaptiCad. Mi agradecimiento también a quienes revisaron la tercera edición: Thomas G. Johnson, *California State University*; Umit Uyar, *City University of New York*; Thomas L. Drake, *Clemson University*; y Richard Molyet, *University of Toledo*. Por último, doy las gracias a mi esposa Sandra por animarme a realizar este proyecto.

M. MORRIS MANO

1 Sistemas binarios

1-1 SISTEMAS DIGITALES

Los sistemas digitales desempeñan un papel tan destacado en la vida cotidiana que el actual periodo tecnológico se conoce como "era digital". Los sistemas digitales se utilizan en comunicaciones, transacciones de negocios, control de tráfico, navegación espacial, tratamiento médico, monitoreo meteorológico, Internet y muchas empresas comerciales, industriales y científicas. Tenemos teléfonos digitales, televisión digital, discos versátiles digitales, cámaras digitales y, desde luego, computadoras digitales. La propiedad más notable de la computadora digital es su generalidad. Es capaz de seguir una secuencia de instrucciones, llamada programa, que opera con ciertos datos. El usuario puede especificar y modificar el programa o los datos según necesidades determinadas. Gracias a esta flexibilidad, las computadoras digitales de uso general son capaces de realizar diversas tareas de procesamiento de información que cubren una amplia gama de aplicaciones.

Una característica de los sistemas digitales es su capacidad para manipular elementos discretos de información. Todo conjunto restringido a un número finito de elementos contiene información discreta. Ejemplos de conjuntos discretos son los 10 dígitos decimales, las 26 letras del alfabeto, los 52 naipes de la baraja común y las 64 casillas de un tablero de ajedrez. Las primeras computadoras digitales se usaron para efectuar cálculos numéricos. En este caso, los elementos discretos que se usaron fueron los dígitos. El término *digital* surgió de esta aplicación. En un sistema digital, los elementos discretos de información se representan mediante cantidades físicas llamadas señales. Las más comunes son señales eléctricas, como voltajes y corrientes. En los circuitos que implementan dichas señales predominan los dispositivos electrónicos llamados transistores. En casi todos los sistemas digitales electrónicos actuales, las señales emplean sólo dos valores discretos, por lo que decimos que son *binarios*. Un dígito binario, llamado *bit*, tiene dos valores: 0 y 1. Los elementos discretos de información se representan con grupos de bits llamados *códigos binarios*. Por ejemplo, los dígitos decimales 0 a 9 se representan en un sistema digital con un código de cuatro bits. Mediante el uso de diversas técnicas, es posible hacer que los grupos de bits representen símbolos discretos, y luego usar-

los para desarrollar el sistema en un formato digital. Así, un sistema digital es un sistema que manipula elementos discretos de información representados internamente en forma binaria.

Las cantidades discretas de información podrían surgir de la naturaleza de los datos procesados, o bien cuantizarse a partir de un proceso continuo. Por ejemplo, una lista de nómina es un proceso inherentemente discreto que contiene nombres de empleados, números de seguro social, salarios quincenales, impuestos sobre la renta, etcétera. El cheque de paga de un empleado se procesa empleando valores discretos de datos como las letras del alfabeto (nombres), los dígitos (salario) y símbolos especiales (como $). Por otra parte, un investigador científico podría observar un proceso continuo, pero registrar únicamente cantidades específicas en forma tabular. Así, el científico está cuantizando sus datos continuos, haciendo que cada cifra de su tabla sea una cantidad discreta. En muchos casos, un convertidor analógico a digital puede efectuar automáticamente la cuantización de un proceso.

La computadora digital de uso general es el ejemplo más conocido de un sistema digital. Las partes principales de una computadora son la unidad de memoria, la unidad central de procesamiento y las unidades de entrada y salida. La unidad de memoria guarda los programas, entradas, salidas y datos intermedios. La unidad central de procesamiento realiza operaciones aritméticas y otras operaciones de procesamiento de datos especificadas por el programa. El programa y los datos que el usuario preparó se transfieren a la memoria mediante un dispositivo de entrada, como un teclado. Un dispositivo de salida, por ejemplo, una impresora, recibe los resultados de los cálculos y presenta al usuario los resultados impresos. Las computadoras digitales pueden manejar muchos dispositivos de entrada y salida. Uno muy útil es la unidad de comunicación que permite interactuar con otros usuarios a través de Internet. Las computadoras digitales son instrumentos potentes capaces de efectuar no sólo cálculos aritméticos sino también operaciones lógicas. Además, se les puede programar para que tomen decisiones con base en condiciones internas y externas.

Hay razones de peso para incluir circuitos digitales en productos comerciales. Al igual que las computadoras digitales, casi todos los dispositivos digitales son programables. Si modificamos el programa de un dispositivo programable, podremos usar el mismo hardware para muchas aplicaciones distintas. El costo de los dispositivos digitales ha bajado drásticamente gracias a los adelantos en la tecnología de los circuitos integrados digitales. A medida que aumenta el número de transistores que es posible incluir en un trozo de silicio, a fin de producir funciones complejas, el costo por unidad baja y el precio de los dispositivos digitales se reduce. Los equipos construidos con circuitos integrados digitales pueden efectuar cientos de millones de operaciones por segundo. Es posible extremar la fiabilidad con que operan los sistemas digitales empleando códigos de corrección de errores. Un ejemplo de esto es el disco digital versátil (DVD, *digital versatile disk*), en el que se graba información digital que contiene vídeo, audio y otros tipos de datos, sin perder un solo elemento. La información digital de un DVD se graba de forma tal que, al examinarse el código de cada muestra digital antes de reproducir su información, es posible identificar y corregir automáticamente cualquier error.

Un sistema digital es una interconexión de módulos digitales. Si queremos entender cómo funciona cada módulo digital, necesitaremos conocimientos básicos de circuitos digitales y de su función lógica. Los primeros siete capítulos del libro presentan las herramientas básicas del diseño digital, como las estructuras de compuertas lógicas, los circuitos combinacionales y secuenciales, y los dispositivos lógicos programables. El capítulo 8 presenta el diseño digital en el nivel de transferencia de registros (RTL, *register transfer level*). Los capítulos 9 y 10 se ocupan de los circuitos secuenciales asincrónicos y de las diversas familias de lógica digital integrada. Los capítulos 11 y 12 presentan los circuitos integrados comerciales y muestran cómo se pueden conectar en el laboratorio para realizar experimentos con circuitos digitales.

Una tendencia importante en el campo del diseño digital es el uso del lenguaje de descripción de hardware (HDL, *hardware description language*). HDL se parece a los lenguajes de programación y permite describir circuitos digitales en forma textual. Sirve para simular sistemas digitales y verificar su funcionamiento antes de crearlos en hardware. También se utiliza junto con herramientas de síntesis lógica para automatizar el diseño. Presentaremos en todo el libro descripciones de circuitos digitales en HDL.

Como ya se explicó antes, los sistemas digitales manipulan cantidades discretas de información que se representan en forma binaria. Los operandos de los cálculos podrían expresarse en el sistema numérico binario. Otros elementos discretos, como los dígitos decimales, se representan con códigos binarios. Los datos se procesan empleando señales binarias manipuladas por elementos lógicos binarios. Las cantidades se guardan en elementos de almacenamiento binarios. El objetivo de este capítulo es presentar los diversos conceptos binarios como marco de referencia para los temas que se estudiarán en los capítulos siguientes.

1-2 NÚMEROS BINARIOS

Un número decimal, como 7,392, representa una cantidad igual a 7 millares más 3 centenas, más 9 decenas, más 2 unidades. Los millares, centenas, etcétera, son potencias de 10 que están implícitas en la posición de los coeficientes. Si queremos ser más exactos, deberíamos escribir 7,392 así:

$$7 \times 10^3 + 3 \times 10^2 + 9 \times 10^1 + 2 \times 10^0$$

No obstante, por convención, se escriben únicamente los coeficientes y se deducen las potencias necesarias de 10 de la posición que dichos coeficientes ocupan. En general, un número con punto decimal se representa con una serie de coeficientes, así:

$$a_5 a_4 a_3 a_2 a_1 a_0 \cdot a_{-1} a_{-2} a_{-3}$$

Los coeficientes a_j son cualesquiera de los 10 dígitos (0, 1, 2, . . . , 9); el valor del subíndice j indica el valor de posición y, por tanto, la potencia de 10 por la que se deberá multiplicar ese coeficiente. Esto puede expresarse así:

$$10^5 a_5 + 10^4 a_4 + 10^3 a_3 + 10^2 a_2 + 10^1 a_1 + 10^0 a_0 + 10^{-1} a_{-1} + 10^{-2} a_{-2} + 10^{-3} a_{-3}$$

Decimos que el sistema numérico decimal es *base* 10 porque usa 10 dígitos y los coeficientes se multiplican por potencias de 10. El sistema *binario* es un sistema numérico diferente. Sus coeficientes sólo pueden tener dos valores: 0 o 1. Cada coeficiente a_j se multiplica por 2^j. Por ejemplo, el equivalente decimal del número binario 11010.11 es 26.75, como puede verse si multiplicamos los coeficientes por potencias de 2:

$$1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} = 26.75$$

En general, un número expresado en un sistema base r consiste en coeficientes que se multiplican por potencias de r:

$$a_n \cdot r^n + a_{n-1} \cdot r^{n-1} + \ldots + a_2 \cdot r^2 + a_1 \cdot r + a_0 + a_{-1} \cdot r^{-1} + a_{-2} \cdot r^{-2} + \ldots + a_{-m} \cdot r^{-m}$$

El valor de los coeficientes a_j varía entre 0 y $r - 1$. Para distinguir entre números con diferente base, encerramos los coeficientes en paréntesis y añadimos un subíndice que indica la base empleada (aunque a veces se hace una excepción en el caso de los números decimales, si por el contexto es obvio que la base es 10). Un ejemplo de número base 5 es

$$(4021.2)_5 = 4 \times 5^3 + 0 \times 5^2 + 2 \times 5^1 + 1 \times 5^0 + 2 \times 5^{-1} = (511.4)_{10}$$

Los valores de los coeficientes en base 5 sólo pueden ser 0, 1, 2, 3 y 4. El sistema numérico octal es un sistema base 8 que tiene ocho dígitos: 0, 1, 2, 3, 4, 5, 6 y 7. Un ejemplo de número octal es 127.4. Para determinar su valor decimal equivalente, expandimos el número como una serie de potencias con base 8:

$$(127.4)_8 = 1 \times 8^2 + 2 \times 8^1 + 7 \times 8^0 + 4 \times 8^{-1} = (87.5)_{10}$$

Advierta que los dígitos 8 y 9 no pueden aparecer en un número octal.

Se acostumbra tomar del sistema decimal los r dígitos requeridos si la base del número es menor que 10, y utilizar las letras del alfabeto para complementar los 10 dígitos decimales si la base del número es mayor que 10. Por ejemplo, en el sistema numérico *hexadecimal* (base 16), los primeros 10 dígitos se toman del sistema decimal, y se usan las letras A, B, C, D, E y F para los dígitos 10, 11, 12, 13, 14 y 15, respectivamente. He aquí un ejemplo de número hexadecimal:

$$(B65F)_{16} = 11 \times 16^3 + 6 \times 16^2 + 5 \times 16^1 + 15 \times 16^0 = (46{,}687)_{10}$$

Como ya se señaló, los dígitos de los números binarios se llaman *bits*. Si un bit es igual a 0, no contribuye a la suma durante la conversión. Por tanto, la conversión de binario a decimal puede efectuarse sumando los números con potencias de 2 correspondientes a los bits que son 1. Por ejemplo,

$$(110101)_2 = 32 + 16 + 4 + 1 = (53)_{10}$$

Este número binario tiene cuatro unos. El número decimal equivalente es la suma de las cuatro potencias de 2 correspondientes. En la tabla 1-1 se presentan los primeros 24 números que se obtienen al elevar 2 a la n potencia. En computación, llamamos K (kilo) a 2^{10}, M (mega) a 2^{20}, G (giga) a 2^{30} y T (tera) a 2^{40}. Así, $4K = 2^{12} = 4096$ y $16M = 2^{24} = 16{,}777{,}216$. La capacidad de las computadoras por lo regular se da en bytes. Un *byte* es igual a ocho bits y puede representar un carácter del teclado. Un disco duro para computadora con capacidad de 4 gigabytes puede almacenar $4G = 2^{32}$ bytes (aproximadamente 4,000 millones de bytes).

Las operaciones aritméticas con números base r siguen las mismas reglas que los números decimales. Cuando se utiliza una base distinta de la conocida base 10, hay que tener cuidado

Tabla 1-1
Potencias de dos

n	2n	n	2n	n	2n
0	1	8	256	16	65,536
1	2	9	512	17	131,072
2	4	10	1,024	18	262,144
3	8	11	2,048	19	524,288
4	16	12	4,096	20	1,048,576
5	32	13	8,192	21	2,097,152
6	64	14	16,384	22	4,194,304
7	128	15	32,768	23	8,388,608

de usar únicamente los *r* dígitos permitidos. He aquí ejemplos de suma, resta y multiplicación de dos números binarios:

sumando:	101101	minuendo:	101101	multiplicando:	1011
sumando:	+100111	sustraendo:	−100111	multiplicador:	× 101
suma:	1010100	diferencia:	000110		1011
					0000
					1011
				producto:	110111

La suma de dos números binarios se calcula empleando las mismas reglas de la suma decimal, excepto que los dígitos de la suma en toda posición significativa sólo pueden ser 0 o 1. Cualquier acarreo que se genere en una posición significativa dada se sumará al par de dígitos que está en la siguiente posición significativa más alta. La resta es un poco más complicada. Las reglas siguen siendo las de la resta decimal, sólo que el préstamo en una posición significativa dada suma 2 al dígito del minuendo. (En el sistema decimal, un préstamo suma 10 al dígito del minuendo.) La multiplicación es muy sencilla. Los dígitos del multiplicador siempre son 1 o 0; por tanto, los productos parciales o bien son iguales al multiplicando, o son 0.

1-3 CONVERSIONES DE BASE NUMÉRICA

La conversión de un número base *r* a decimal se efectúa expandiendo el número a una serie de potencias y sumando todos los términos, como ya se explicó. A continuación presentaremos un procedimiento general para la operación inversa de convertir un número decimal en un número base *r*. Si el número lleva punto, será necesario separar la parte entera de la parte fraccionaria, pues cada parte se convierte de manera distinta. La conversión de un entero decimal en un número base *r* se efectúa dividiendo el número y todos sus cocientes sucesivos entre *r* y acumulando los residuos. La mejor forma de explicar el procedimiento es con un ejemplo.

EJEMPLO 1-1

Convertir 41 decimal a binario. Primero, se divide 41 entre 2 para dar un cociente entero de 20 y un residuo de $\frac{1}{2}$. Se vuelve a dividir el cociente entre 2 para dar un nuevo cociente y un nuevo residuo. El proceso se continúa hasta que el cociente entero es 0. Los *coeficientes* del número binario deseado se obtienen a partir de los *residuos*, como sigue:

	Cociente entero		**Residuo**	**Coeficiente**
$41/2 =$	20	+	$\frac{1}{2}$	$a_0 = 1$
$20/2 =$	10	+	0	$a_1 = 0$
$10/2 =$	5	+	0	$a_2 = 0$
$5/2 =$	2	+	$\frac{1}{2}$	$a_3 = 1$
$2/2 =$	1	+	0	$a_4 = 0$
$1/2 =$	0	+	$\frac{1}{2}$	$a_5 = 1$

Por tanto, la respuesta es $(41)_{10} = (a_5a_4a_3a_2a_1a_0)_2 = (101001)_2$

El proceso aritmético se puede plantear de forma más conveniente como sigue:

Entero	Residuo
41	
20	1
10	0
5	0
2	1
1	0
0	1

101001 = respuesta

La conversión de enteros decimales a cualquier sistema base r es similar a este ejemplo, sólo que se divide entre r en vez de entre 2.

EJEMPLO 1-2

Convertir 153 decimal a octal. La base r en este caso es 8. Primero dividimos 153 entre 8 para obtener un cociente entero de 19 y un residuo de 1. Luego dividimos 19 entre 8 para obtener un cociente entero de 2 y un residuo de 3. Por último, dividimos 2 entre 8 para obtener un cociente de 0 y un residuo de 2. Este proceso se puede plantear así:

153	
19	1
2	3
0	2

$= (231)_8$

La conversión de una *fracción* decimal a binario se efectúa con un método similar al que se utiliza con enteros, pero se multiplica en lugar de dividir y se acumulan enteros en vez de residuos. En este caso, también, la mejor explicación es un ejemplo.

EJEMPLO 1-3

Convertir $(0.6875)_{10}$ a binario. Primero, multiplicamos 0.6875 por 2 para obtener un entero y una fracción. La nueva fracción se multiplica por 2 para dar un nuevo entero y una nueva fracción. El proceso se continúa hasta que la fracción es 0 o hasta que se tienen suficientes dígitos para la precisión deseada. Los coeficientes del número binario se obtienen de los enteros, así:

	Entero		Fracción	Coeficiente
$0.6875 \times 2 =$	1	+	0.3750	$a_{-1} = 1$
$0.3750 \times 2 =$	0	+	0.7500	$a_{-2} = 0$
$0.7500 \times 2 =$	1	+	0.5000	$a_{-3} = 1$
$0.5000 \times 2 =$	1	+	0.0000	$a_{-4} = 1$

Por tanto, la respuesta es $(0.6875)_{10} = (0.a_{-1}a_{-2}a_{-3}a_{-4})_2 = (0.1011)_2$

Para convertir una fracción decimal a un número expresado en base r, seguimos un procedimiento similar, multiplicando por r en vez de por 2. Los coeficientes obtenidos a partir de los enteros tendrán valores entre 0 y $r - 1$, en vez de ser sólo 0 y 1.

■

EJEMPLO 1-4

Convertir $(0.513)_{10}$ a octal.

$$0.513 \times 8 = 4.104$$
$$0.104 \times 8 = 0.832$$
$$0.832 \times 8 = 6.656$$
$$0.656 \times 8 = 5.248$$
$$0.248 \times 8 = 1.984$$
$$0.984 \times 8 = 7.872$$

La respuesta, con siete cifras significativas, se obtiene de la parte entera de los productos

$$(0.513)_{10} = (0.406517\dots)_8$$

La conversión de números decimales que tienen tanto parte entera como parte fraccionaria se efectúa convirtiendo por separado las dos partes y combinando después las dos respuestas. Si usamos los resultados de los ejemplos 1-1 y 1-3, obtendremos

$$(41.6875)_{10} = (101001.1011)_2$$

De los ejemplos 1-2 y 1-4 tenemos

$$(153.513)_{10} = (231.406517)_8$$

■

1-4 NÚMEROS OCTALES Y HEXADECIMALES

Las conversiones entre binario, octal y hexadecimal desempeñan un papel importante en las computadoras digitales. Puesto que $2^3 = 8$ y $2^4 = 16$, cada dígito octal corresponde a tres dígitos binarios y cada dígito hexadecimal corresponde a cuatro dígitos binarios. En la tabla 1-2 se presentan los primeros 16 números de los sistemas numéricos decimal, binario, octal y hexadecimal.

La conversión de binario a octal se efectúa fácilmente acomodando los dígitos del número binario en grupos de tres, partiendo del punto binario tanto a la izquierda como a la derecha. Luego, se asigna el dígito octal correspondiente a cada grupo. Este ejemplo ilustra el procedimiento:

$$\begin{array}{ccccccccccc} (\,10 & 110 & 001 & 101 & 011 & \cdot & 111 & 100 & 000 & 110\,)_2 & = (26153.7460)_8 \\ 2 & 6 & 1 & 5 & 3 & & 7 & 4 & 0 & 6 & \end{array}$$

Tabla 1-2
Números con diferente base

Decimal (base 10)	Binario (base 2)	Octal (base 8)	Hexadecimal (base 16)
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

La conversión de binario a hexadecimal es similar, sólo que el número binario se divide en grupos de cuatro dígitos:

$$\begin{array}{cccccccl} (10 & 1100 & 0110 & 1011 & \cdot & 1111 & 0010)_2 & = (2C6B.F2)_{16} \\ 2 & C & 6 & B & & F & 2 & \end{array}$$

Es fácil recordar el dígito hexadecimal (u octal) que corresponde a cada grupo de dígitos binarios si se examinan los valores de la tabla 1-2.

La conversión de octal o hexadecimal a binario se hace invirtiendo el procedimiento anterior. Cada dígito octal se convierte a su equivalente binario de tres dígitos. Asimismo, cada dígito hexadecimal se convierte en su equivalente binario de cuatro dígitos. Los ejemplos siguientes ilustran el procededimiento:

$$\begin{array}{lccccccc} (673.124)_8 = (& 110 & 111 & 011 & \cdot & 001 & 010 & 100\ )_2 \\ & 6 & 7 & 3 & & 1 & 2 & 4 \end{array}$$

y

$$\begin{array}{lccccc} (306.D)_{16} = (& 0011 & 0000 & 0110 & \cdot & 1101\ )_2 \\ & 3 & 0 & 6 & & D \end{array}$$

Es difícil trabajar con números binarios porque requieren tres o cuatro veces más dígitos que sus equivalentes decimales. Por ejemplo, el número binario 111111111111 equivale al 4095 decimal. No obstante, las computadoras digitales emplean números binarios y hay ocasiones en que el operador o usuario humano necesita comunicarse directamente con la máquina empleando números binarios. Un esquema que retiene el sistema binario en la computadora, pero reduce el número de dígitos que la persona debe considerar, aprovecha la relación entre el sistema

numérico binario y los sistemas octal y hexadecimal. Con ese método, la persona piensa en términos de números octales o hexadecimales y efectúa la conversión requerida por inspección cuando se hace necesaria la comunicación directa con la máquina. Así, el número binario 111111111111 tiene 12 dígitos y se expresa en octal como 7777 (cuatro dígitos) y en hexadecimal como FFF (tres dígitos). Cuando dos personas hablan entre sí (acerca de números binarios en la computadora), la representación octal o hexadecimal es más deseable porque se puede expresar de manera más compacta con una tercera o cuarta parte de los dígitos requeridos para el número binario equivalente. Por ello, casi todos los manuales de computadora utilizan números octales o hexadecimales para especificar cantidades binarias. La selección de cualquiera de estos dos sistemas es arbitraria, aunque se suele preferir el hexadecimal porque puede representar un byte con dos dígitos.

1-5 COMPLEMENTOS

En las computadoras digitales se usan complementos para simplificar la operación de resta y para efectuar manipulaciones lógicas. Hay dos tipos de complementos para cada sistema base r: el complemento a la base y el complemento a la base disminuida. El primero se denomina complemento a r, mientras que el segundo es el complemento a $(r - 1)$. Si sustituimos el valor de la base r en estos nombres, los dos tipos son el complemento a dos y el complemento a uno, en el caso de los números binarios, y el complemento a diez y el complemento a nueve en el caso de los números decimales.

Complemento a la base disminuida

Dado un número N en base r que tiene n dígitos, el complemento a $(r - 1)$ de N se define como $(r^n - 1) - N$. En el caso de números decimales, $r = 10$ y $r - 1 = 9$, así que el complemento a nueve de N es $(10^n - 1) - N$. En este caso, 10^n representa un número que consiste en un uno seguido de n ceros. $10^n - 1$ es un número representado por n nueves. Por ejemplo, si $n = 4$, tenemos $10^4 = 10{,}000$ y $10^4 - 1 = 9999$. De esto se sigue que el complemento a nueve de un número decimal se obtiene restando cada dígito a nueve. He aquí algunos ejemplos numéricos:

El complemento a nueve de 546700 es $999999 - 546700 = 453299$.

El complemento a nueve de 012398 es $999999 - 012398 = 987601$.

En el caso de los números binarios, $r = 2$ y $r - 1 = 1$, así que el complemento a uno de N es $(2^n - 1) - N$. Aquí también, 2^n se representa con un número binario que consiste en un uno seguido de n ceros. $2^n - 1$ es un número binario representado por n unos. Por ejemplo, si $n = 4$, tenemos $2^4 = (10000)_2$ y $2^4 - 1 = (1111)_2$. Así, el complemento a uno de un número binario se obtiene restando cada dígito a uno. Sin embargo, al restar dígitos binarios a 1 podemos tener $1 - 0 = 1$ o bien $1 - 1 = 0$, lo que hace que el bit cambie de 0 a 1 o de 1 a 0. Por tanto, el complemento a uno de un número binario se forma cambiando los unos a ceros y los ceros a unos. He aquí algunos ejemplos numéricos:

El complemento a uno de 1011000 es 0100111.

El complemento a uno de 0101101 es 1010010.

El complemento a $(r - 1)$ de los números octales y hexadecimales se obtiene restando cada dígito a 7 y F (15 decimal), respectivamente.

Complemento a la base

El complemento a r de un número N de n dígitos en base r se define como $r^n - N$, para $N \neq 0$, y 0 para $N = 0$. Si comparamos con el complemento a $(r - 1)$, veremos que el complemento a r se obtiene sumando 1 al complemento a $(r - 1)$, ya que $r^n - N = [(r^n - 1) - N] + 1$. Así pues, el complemento a 10 del número decimal 2389 es $7610 + 1 = 7611$, y se obtiene sumando 1 al valor del complemento a nueve. El complemento a dos del número binario 101100 es $010011 + 1 = 010100$, y se obtiene sumando 1 al valor del complemento a uno.

Puesto que 10^n es un número que se representa con un uno seguido de n ceros, $10^n - N$, que es el complemento a 10 de N, también puede formarse dejando como están todos los ceros menos significativos, restando a 10 el primer dígito menos significativo distinto de cero, y restando a 9 los demás dígitos a la izquierda.

El complemento a 10 de 012398 es 987602.

El complemento a 10 de 246700 es 753300.

El complemento a 10 del primer número se obtiene restando 8 a 10 en la posición menos significativa y restando a 9 todos los demás dígitos. El complemento a 10 del segundo número se obtiene dejando como están los dos ceros de la derecha, restando 7 a 10 y restando a 9 los otros tres dígitos.

De forma similar, el complemento a dos se forma dejando como están todos los ceros menos significativos y el primer uno, y sustituyendo los unos por ceros y los ceros por unos en las demás posiciones a la izquierda.

El complemento a dos de 1101100 es 0010100.

El complemento a dos de 0110111 es 1001001.

El complemento a dos del primer número se obtiene dejando como están los dos ceros menos significativos y el primer uno, y sustituyendo después los unos por ceros y los ceros por unos en las cuatro posiciones más significativas. El complemento a dos del segundo número se obtiene dejando como está el uno menos significativo y complementando todos los demás dígitos a la izquierda.

En las definiciones anteriores se supuso que los números no llevan punto. Si el número N original lleva punto, deberá quitarse temporalmente para formar el complemento a r o a $(r - 1)$, y volver a colocarlo después en el número complementado en la misma posición relativa. También vale la pena mencionar que el complemento del complemento restablece el valor original del número. El complemento a r de N es $r^n - N$. El complemento del complemento es $r^n - (r^n - N) = N$, o sea, el número original.

Resta con complementos

El método directo que se enseña en la escuela primaria para restar utiliza el concepto de préstamo. Pedimos prestado un uno de la siguiente posición más significativa cuando el dígito del minuendo es menor que el del sustraendo. El método funciona bien cuando se resta con lápiz y papel, pero cuando la resta se implementa en hardware digital el método es menos eficiente que si se usan complementos.

La resta de dos números de n dígitos sin signo, $M - N$, en base r se efectúa así:

1. Sume el minuendo, M, al complemento a r del sustraendo, N. Esto da $M + (r^n - N) = M - N + r^n$.
2. Si $M \geq N$, la suma producirá un acarreo final, r^n, que puede desecharse; lo que queda es el resultado $M - N$.
3. Si $M < N$, la suma no produce un acarreo final y es igual a $r^n - (N - M)$, que es el complemento a r de $(N - M)$. Para obtener la respuesta en una forma conocida, se toma el complemento a r de la suma y se le antepone un signo de menos.

Los ejemplos que siguen ilustran el procedimiento:

EJEMPLO 1-5

Utilizando complemento a 10, restar $72532 - 3250$.

$$
\begin{aligned}
M &= \ 72532 \\
\text{Complemento a 10 de } N &= +\ \underline{96750} \\
\text{Suma} &= 169282 \\
\text{Desechar acarreo final } 10^5 &= -\underline{100000} \\
\textit{Respuesta} &= \ 69282
\end{aligned}
$$

■

Observe que M tiene cinco dígitos y N sólo tiene cuatro. Ambos números deben tener el mismo número de dígitos, así que escribimos N como 03250. La obtención del complemento a 10 de N produce un nueve en la posición más significativa. El acarreo final indica que $M \geq N$ y que el resultado es positivo.

EJEMPLO 1-6

Utilizando complemento a 10, restar $3250 - 72532$.

$$
\begin{aligned}
M &= \ 03250 \\
\text{Complemento a 10 de } N &= +\ \underline{27468} \\
\text{Suma} &= \ 30718
\end{aligned}
$$

No hay acarreo final.
Por tanto, la respuesta es $-$(complemento a 10 de 30718) $= -69282$

■

Cabe señalar que, dado que $3250 < 72532$, el resultado es negativo. Puesto que estamos manejando números sin signo, en realidad es imposible obtener un resultado sin signo para este caso. Al restar con complementos, la respuesta negativa se reconoce por la ausencia de acarreo final y por el resultado complementado. Cuando trabajamos con lápiz y papel, podemos convertir la respuesta en un número negativo con signo y así expresarlo en una forma conocida.

La resta con complementos es similar en el caso de los números binarios, y se usa el procedimiento ya expuesto.

EJEMPLO 1-7

Dados los números binarios $X = 1010100$ y $Y = 1000011$, realizar las restas **a)** $X - Y$ y **b)** $Y - X$ empleando complementos a dos.

a)

$$\begin{aligned} X &= 1010100 \\ \text{Complemento a dos de } Y &= +\underline{0111101} \\ \text{Suma} &= 10010001 \\ \text{Desechar acarreo final } 2^7 &= -\underline{10000000} \\ \textit{Respuesta}:\ X - Y &= 0010001 \end{aligned}$$

b)

$$\begin{aligned} Y &= 1000011 \\ \text{Complemento a dos de } X &= +\underline{0101100} \\ \text{Suma} &= 1101111 \end{aligned}$$

No hay acarreo final.
Por tanto, la respuesta es $Y - X = -$(complemento a dos de 1101111) $= -0010001$

La resta de números sin signo también se puede efectuar usando el complemento a $(r - 1)$. Recordemos que el complemento a $(r - 1)$ es uno menos que el complemento a r. Por ello, el resultado de sumar el minuendo al complemento del sustraendo produce una suma uno menos que la diferencia correcta cuando hay acarreo final. Quitar el acarreo final y sumar 1 a la suma se denomina *acarreo circular*.

EJEMPLO 1-8

Repetir el ejemplo 1-7 empleando complemento a uno.

a) $X - Y = 1010100 - 1000011$

$$\begin{aligned} X &= 1010100 \\ \text{Complemento a uno de } Y &= +\underline{0111100} \\ \text{Suma} &= 10010000 \\ \text{Acarreo circular} &= +\underline{1} \\ \textit{Respuesta}:\ X - Y &= 0010001 \end{aligned}$$

b) $Y - X = 1000011 - 1010100$

$$\begin{aligned} Y &= 1000011 \\ \text{Complemento a uno de } X &= +\underline{0101011} \\ \text{Suma} &= 1101110 \end{aligned}$$

No hay acarreo final.
Por tanto, la respuesta es $Y - X = -$(complemento a uno de 1101110) $= -0010001$

Observe que el resultado negativo se obtiene tomando el complemento a uno de la suma, ya que éste es el tipo de complemento empleado. El procedimiento con acarreo circular también es válido para restar números decimales sin signo, utilizando complemento a nueve.

1-6 NÚMEROS BINARIOS CON SIGNO

Los enteros positivos (incluido el cero) se representan como números sin signo. Sin embargo, para representar enteros negativos se necesita una notación que distinga a los valores negativos. En la aritmética ordinaria, indicamos un número negativo con un signo de menos, y uno positivo, con un signo de más. Por limitaciones del hardware, las computadoras deben representar todo con dígitos binarios. Se acostumbra representar el signo con un bit colocado en la posición extrema izquierda del número. La convención es que el bit sea cero si el número es positivo, y uno si es negativo.

Es importante darse cuenta de que los números binarios, tanto con signo como sin él, se representan en las computadoras con una cadena de bits. El usuario determina si el número tiene signo o no. Si el número binario posee signo, el bit de la extrema izquierda representará el signo y el resto de los bits representarán el número. Si se supone que el número binario carece de signo, el bit de la extrema izquierda será el bit más significativo del número. Por ejemplo, la cadena de bits 01001 se considera como 9 (binario sin signo) o +9 (binario con signo), porque el bit de la extrema izquierda es cero. La cadena de bits 11001 representa el equivalente binario de 25 cuando se le considera un número sin signo, o −9 cuando se le considera un número con signo. Ello se debe a que el uno de la posición extrema izquierda indica que el número es negativo, y los otros cuatro bits representan 9 en binario. Normalmente, no hay problema para identificar los bits si se conoce con antelación el tipo de representación del número.

La representación de los números con signo de nuestro último ejemplo usa la convención de *magnitud con signo*. En esta notación, el número consiste en una magnitud y un símbolo (+ o −) o un bit (0 o 1) que indica el signo. Ésta es la representación de números con signo que se emplea en la aritmética ordinaria. Al implementar operaciones aritméticas en una computadora, es más conveniente usar un sistema distinto para representar números negativos, denominado sistema de *complemento con signo*. En este sistema, los números negativos se indican con su complemento. Mientras que el sistema de magnitud con signo hace negativo a un número cambiando su signo, el sistema de complemento con signo hace negativo a un número convirtiéndolo en su complemento. Puesto que los números positivos siempre inician con cero (más) en la posición de extrema izquierda, el complemento siempre iniciará con uno, lo que indica un número negativo. El sistema de complemento con signo puede utilizar el complemento a uno o a dos, aunque este último es el más común.

Por ejemplo, considere el número 9 representado en binario con ocho bits. +9 se representa con un bit de signo cero en la posición de extrema izquierda, seguido del equivalente binario de 9, lo que da 00001001. Cabe señalar que los ocho bits deben tener valor, por lo que se insertan ceros después del bit de signo, hasta el primer uno. Aunque sólo hay una forma de representar +9, hay tres formas de representar −9 con ocho bits:

representación de magnitud con signo:	10001001
representación de complemento a uno con signo:	11110110
representación de complemento a dos con signo:	11110111

En el sistema de magnitud con signo, se obtiene −9 a partir de +9 cambiando el bit de signo en la posición de extrema izquierda, de cero a uno. En complemento a uno con signo, se ob-

Tabla 1-3
Números binarios con signo

Decimal	Complemento a dos con signo	Complemento a uno con signo	Magnitud con signo
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
−0	—	1111	1000
−1	1111	1110	1001
−2	1110	1101	1010
−3	1101	1100	1011
−4	1100	1011	1100
−5	1011	1010	1101
−6	1010	1001	1110
−7	1001	1000	1111
−8	1000	—	—

tiene −9 complementando todos los bits de +9, incluido el bit de signo. La representación de −9 en complemento a dos con signo se obtiene tomando el complemento a dos del número positivo, incluido el bit de signo.

En la tabla 1-3 se muestran todos los números binarios de cuatro bits con signo en las tres representaciones. También se ha incluido el número decimal equivalente como referencia. Observe que los números positivos son idénticos en las tres representaciones, y tienen un cero en la posición de extrema izquierda. El sistema de complemento a dos con signo sólo tiene una forma de representar el cero, que siempre es positivo. Los otros dos sistemas tienen un cero positivo y un cero negativo, cosa que no se usa en la aritmética ordinaria. Advierta que todos los números negativos tienen un uno en la posición de bit de extrema izquierda; así es como se distinguen de los números positivos. Es posible representar 16 números binarios con cuatro bits. En las representaciones de magnitud con signo y complemento a uno con signo, hay ocho números positivos y ocho números negativos, incluidos dos ceros. En la representación de complemento a dos, hay ocho números positivos, que incluyen un cero, y ocho números negativos.

El sistema de magnitud con signo se usa en aritmética ordinaria, pero resulta torpe en aritmética de computadoras porque hay que manejar por separado el signo y la magnitud. Es por ello que normalmente se usa el complemento con signo. El complemento a uno conlleva ciertas dificultades y casi nunca se emplea en operaciones aritméticas. Es útil como operación lógica porque el cambio de uno a cero o de cero a uno equivale a una operación de complemento lógico, como se verá en el siguiente capítulo. La explicación que sigue de la aritmética binaria con signo se ocupa exclusivamente de la representación de números negativos como complemento a dos con signo. Se pueden aplicar los mismos procedimientos al sistema de complemento a uno con signo si se incluye el acarreo circular, como se hizo en el caso de los números sin signo.

Suma aritmética

La suma de dos números en el sistema de magnitud con signo sigue las reglas de la aritmética ordinaria. Si los signos son iguales, sumamos las dos magnitudes y asignamos ese signo a la suma. Si los signos son distintos, se resta la magnitud menor a la mayor y se asigna al resultado el signo de la magnitud mayor. Por ejemplo, $(+25) + (-37) = -(37 - 25) = -12$, y lo hacemos restando la magnitud menor, 25, a la magnitud mayor, 37, y usando el signo del 37 en el resultado. Este proceso requiere comparar los signos y las magnitudes para decidir entre sumar o restar. El mismo procedimiento se usa con números binarios representados como magnitud con signo. En contraste, la regla para sumar números en el sistema de complemento con signo no requiere comparar ni restar, sólo sumar. El procedimiento es muy sencillo y puede plantearse como sigue para los números binarios:

La suma de dos números binarios con signo, representando los números negativos en forma de complemento a dos con signo, se obtiene sumando los dos números, incluidos los bits de signo. Si se genera un acarreo en la posición de bit del signo, se desecha.

He aquí ejemplos numéricos de sumas:

+ 6	00000110	− 6	11111010
+13	00001101	+13	00001101
+19	00010011	+ 7	00000111
+ 6	00000110	− 6	11111010
−13	11110011	−13	11110011
− 7	11111001	−19	11101101

Tome en cuenta que los números negativos deben estar inicialmente como complemento a dos, y que si la suma obtenida es negativa, está en forma de complemento a dos.

En los cuatro casos, la operación efectuada es suma, e incluye el bit de signo. Cualquier acarreo generado en la posición de bit del signo se desecha, y los resultados negativos están automáticamente en forma de complemento a dos.

Para obtener una respuesta correcta, hay que cuidar que el resultado tenga suficientes bits para dar cabida a la suma. Si partimos con dos números de n bits, y la suma ocupa $n + 1$ bits, decimos que hay un desbordamiento. Cuando efectuamos la suma con lápiz y papel, los desbordamientos no representan un problema, porque no estamos limitados por la anchura de la página. Simplemente añadimos otro 0 a un número positivo u otro 1 a un número negativo en la posición más significativa, a fin de extenderlos a $n + 1$ bits, y realizamos la suma. En las computadoras el desbordamiento sí es un problema porque los números se almacenan en un número finito de bits, y si el resultado excede ese número finito en 1, no cabrá.

La representación de números negativos como complementos resulta extraña para quienes están acostumbrados al sistema de magnitud con signo. Para determinar el valor de un número negativo en complemento a dos, es preciso convertirlo en un número positivo para tenerlo en una forma más familiar. Por ejemplo, el número binario con signo 11111001 es negativo porque el bit de la extrema izquierda es 1. Su complemento a dos es 00000111, que es el equivalente binario de $+7$. Por tanto, reconocemos el número negativo original como -7.

Resta aritmética

La resta de dos números binarios con signo, cuando los números negativos se representan como complemento a dos, es muy sencilla y se realiza así:

Obtenemos el complemento a dos del sustraendo (incluido el bit de signo) y lo sumamos al minuendo (incluido el bit de signo). Si se genera un acarreo en la posición de bit del signo, se desecha.

El procedimiento funciona porque una operación de resta se puede convertir en una operación de suma si se cambia el signo del sustraendo. La relación que sigue lo demuestra:

$$(\pm A) - (+B) = (\pm A) + (-B);$$
$$(\pm A) - (-B) = (\pm A) + (+B).$$

La conversión de un número positivo en uno negativo es fácil si se obtiene su complemento a dos. Lo opuesto también se cumple porque el complemento de un número negativo en forma de complemento produce el número positivo equivalente. Considere la resta de $(-6) - (-13) = +7$. En binario con ocho bits, esto se escribe como $(11111010 - 11110011)$. La resta se convierte en suma obteniendo el complemento a dos del sustraendo (-13) para dar $(+13)$. En binario, esto es $11111010 + 00001101 = 100000111$. Al eliminar el acarreo final, obtendremos la respuesta correcta: 00000111 $(+7)$.

Vale la pena señalar que los números binarios en el sistema de complemento con signo se suman y restan siguiendo las mismas reglas básicas de suma y resta que los números sin signo. Por ello, las computadoras sólo necesitan un circuito en hardware para manejar ambos tipos de aritmética. El usuario o programador deberá interpretar de diferente manera los resultados de tales sumas o restas, dependiendo de si supone que los números tienen signo o no.

1-7 CÓDIGOS BINARIOS

Los sistemas digitales emplean señales que tienen dos valores distintos, y elementos de circuito que tienen dos estados estables. Existe una analogía directa entre señales binarias, elementos binarios de circuito y dígitos binarios. Un número binario de n dígitos, por ejemplo, podría representarse con n elementos binarios de circuito, cada uno de los cuales tiene una señal de salida equivalente a 0 o 1. Los sistemas digitales representan y manipulan no sólo números binarios, sino también muchos otros elementos discretos de información. Cualquier elemento discreto de información distinto dentro de un grupo de cantidades se puede representar con un código binario. Los códigos deben estar en binario porque las computadoras sólo pueden almacenar unos y ceros. Debemos entender que los códigos binarios simplemente cambian los símbolos, no el significado de los elementos de información que representan. Si examinamos al azar los bits de una computadora, veremos que en la mayor parte de los casos representan algún tipo de información codificada, no números binarios.

Un código binario de n bits es un grupo de n bits que puede tener hasta 2^n combinaciones distintas de unos y ceros; cada combinación representa un elemento del conjunto que se está codificando. Un conjunto de cuatro elementos se puede codificar con dos bits, y a cada elemento se asignará una de las combinaciones de bits siguientes: 00, 01, 10, 11. Un conjunto de ocho elementos requiere un código de tres bits, y uno de 16 elementos, un código de 4 bits. Las combinaciones de bits de un código de n bits se determinan contando en binario desde 0 hasta $2^n - 1$. Es preciso asignar a cada elemento una combinación distinta de bits; dos elemen-

tos diferentes no pueden representarse con la misma combinación; en tal caso, la asignación del código sería ambigua.

Aunque el número *mínimo* de bits necesarios para codificar 2^n cantidades distintas es n, no hay un número *máximo* de bits que pueda usarse para un código binario. Por ejemplo, los 10 dígitos decimales podrían codificarse con 10 bits, asignando a cada dígito decimal una combinación de nueve ceros y un uno. En este código binario específico, se asignaría al dígito 6 la combinación de bits 0001000000.

Código BCD

Aunque el sistema numérico binario es el más natural para una computadora, casi todas las personas están acostumbradas al sistema decimal. Una forma de salvar esta diferencia es convertir los números decimales a binario, realizar todos los cálculos aritméticos en binario y luego convertir los resultados a decimal. Este método requiere almacenar los números decimales en la computadora de modo que se puedan convertir a binario. Puesto que la computadora sólo acepta valores binarios, es necesario representar los dígitos decimales con un código a base de unos y ceros. También es posible efectuar las operaciones aritméticas directamente con números decimales si están almacenados en forma codificada en la computadora.

Un código binario tendrá algunas combinaciones de bit no asignadas si el número de elementos del conjunto no es un múltiplo de una potencia de 2. Los 10 dígitos decimales son un conjunto así. Un código binario que distinga 10 elementos deberá contener por lo menos cuatro bits, pero seis de las 16 posibles combinaciones quedarán sin asignarse. Es posible idear diferentes códigos binarios para acomodar cuatro bits en 10 combinaciones distintas. El código que más comúnmente se usa para los dígitos decimales es la asignación binaria directa que se presenta en la tabla 1-4. Se llama "decimal codificado en binario" y se le conoce comúnmente como BCD, por sus siglas en inglés. Es posible usar otros códigos decimales, y se presentarán algunos más adelante en esta sección.

La tabla 1-4 presenta el código de 4 bits para cada dígito decimal. Un número con k dígitos decimales requerirá $4k$ bits en BCD. El número decimal 396 se representa en BCD con 12 bits, así: 0011 1001 0110. Cada grupo de cuatro bits representa un número digital. Un núme-

Tabla 1-4
Decimal codificado en binario (BCD)

Símbolo decimal	Dígito BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

ro decimal en BCD sólo es igual a su número binario equivalente si el número está entre 0 y 9. Un número mayor que 10 se ve diferente en BCD que como número binario, aunque ambos consistan en unos y ceros. Además, las combinaciones binarias 1010 a 1111 no se usan y carecen de significado en el código BCD. Considere el número decimal 185 y su valor correspondiente en BCD y binario:

$$(185)_{10} = (0001\ 1000\ 0101)_{BCD} = (10111001)_2$$

El valor en BCD tiene 12 bits, pero el número binario equivalente sólo necesita ocho bits. Es obvio que un número BCD necesita más bits que su valor binario equivalente, pero el uso de números decimales tiene cierta ventaja porque los datos de entrada y salida de las computadoras se generan por y para personas que usan el sistema decimal.

Es importante entender que los números BCD son números decimales, no binarios, aunque se representen con bits. La única diferencia entre un número decimal y un BCD es que los decimales se escriben con los símbolos 0, 1, 2, . . . , 9 y los números BCD usan el código binario 0000, 0001, 0010, . . . , 1001. El valor decimal es exactamente el mismo. El 10 decimal se representa en BCD con ocho bits: 0001 0000, y el 15 decimal, con 0001 0101. Los valores binarios correspondientes son 1010 y 1111, y sólo tienen cuatro bits.

Suma BCD

Considere la suma de dos dígitos decimales en BCD, junto con un posible acarreo de un par de dígitos anteriores, menos significativos. Puesto que ningún dígito es mayor que 9, la suma no puede ser mayor que $9 + 9 + 1 = 19$, donde el 1 que se suma es el acarreo que “se llevaba”. Suponga que se suman los dígitos BCD como si fueran números binarios. La suma binaria producirá un resultado dentro del intervalo de 0 a 19. En binario, dicho intervalo es de 0000 a 10011, pero en BCD es de 0000 a 1 1001, donde el primer 1 es un acarreo y los cuatro bits siguientes son la suma de los dígitos BCD. Si la suma binaria es 1001 o menos (sin acarreo), el dígito BCD correspondiente es correcto. Sin embargo, cuando la suma binaria es 1010 o más, el resultado es un dígito BCD no válido. La suma de $6 = (0110)_2$ a la suma binaria la convierte en el dígito correcto y también produce el acarreo necesario. Ello se debe a que la diferencia entre un acarreo en la posición de bit más significativa de la suma binaria y un acarreo decimal es de $16 - 10 = 6$. Consideremos estas tres sumas BCD:

$$\begin{array}{rrrrrr}
4 & 0100 & 4 & 0100 & 8 & 1000 \\
+\underline{5} & +\underline{0101} & +\underline{8} & +\underline{1000} & +\underline{9} & \underline{1001} \\
9 & 1001 & 12 & 1100 & 17 & 10001 \\
 & & & +\underline{0110} & & +\underline{0110} \\
 & & & 10010 & & 10111
\end{array}$$

En cada caso, los dos dígitos BCD se suman como si fueran dos números binarios. Si la suma binaria es 1010 o más, se le suma 0110 para obtener la suma correcta de dígitos BCD y el acarreo. En el primer ejemplo, la suma es 9 y es la suma correcta de dígitos BCD. En el segundo ejemplo, la suma binaria produce un dígito BCD no válido. La suma de 0110 produce la suma de dígitos BCD correcta, 0010 (2), y un acarreo. En el tercer ejemplo, la suma binaria produce un acarreo. Esta condición se presenta cuando la suma es 16 o más. Aunque los otros cuatro bits son menores que 1001, la suma binaria requiere una corrección debido

al acarreo. Al sumar 0110, se obtiene la suma de dígitos BCD requerida, 0111 (7), y un acarreo BCD.

La suma de dos números BCD de *n* dígitos sin signo se efectúa siguiendo el mismo procedimiento. Consideremos la suma de 184 + 576 = 760 en BCD:

Acarreo BCD	1	1		
	0001	1000	0100	184
	+0101	0111	0110	+576
Suma binaria	0111	10000	1010	
Sumar 6		0110	0110	
Suma BCD	0111	0110	0000	760

El primer par de dígitos BCD (los menos significativos) produce una suma de dígitos BCD de 0000 y un acarreo para el siguiente par de dígitos. El segundo par de dígitos BCD más el acarreo anterior produce una suma de dígitos de 0110 y un acarreo para el siguiente par de dígitos. El tercer par, más el acarreo, produce una suma binaria de 0111 que no requiere corrección.

Aritmética decimal

La representación de números decimales con signo en BCD es similar a la representación de números con signo en binario. Se puede usar el sistema tan conocido de magnitud y signo, o el de complemento y signo. El signo de un número decimal por lo regular se representa con cuatro bits para ajustarse al código de cuatro bits de los dígitos decimales. Se acostumbra designar el signo de más con cuatro ceros, y el menos, con el equivalente BCD de 9, o sea, 1001.

El sistema de magnitud con signo se usa poco en computación. El sistema de complemento con signo puede usar el complemento a nueve o el complemento a 10, pero este último es el más común. Para obtener el complemento a 10 de un número BCD, primero se obtiene el complemento a nueve y luego se suma 1 al dígito menos significativo. El complemento a nueve se calcula restando a 9 cada dígito.

Los procedimientos desarrollados para el sistema de complemento a dos con signo explicados en la sección anterior también son válidos para el sistema de complemento a 10 con signo que se usa con los números decimales. La suma se efectúa sumando todos los dígitos, incluido el dígito del signo, y desechando el acarreo final. Esto supone que todos los números negativos están en forma de complemento a 10. Consideremos la suma (+375) + (−240) = +135, efectuada en el sistema de complemento con signo.

$$
\begin{array}{r}
0\ \ 375 \\
+9\ \ 760 \\
\hline
0\ \ 135
\end{array}
$$

El 9 en la posición extrema izquierda del segundo número representa un signo menos, y 9760 es el complemento a 10 de 0240. Los dos números se suman y se desecha el acarreo final para obtener +135. Desde luego, los números decimales dentro de la computadora deben estar

en BCD, incluidos los dígitos del signo. La suma se efectúa con dígitos BCD, como ya se explicó.

La resta de números decimales, sea sin signo o en el sistema de complemento a 10 con signo, es igual que en el caso binario. Se obtiene el complemento a 10 del sustraendo y se suma al minuendo. Muchas computadoras tienen hardware especial para realizar cálculos aritméticos directamente con números decimales en BCD. El usuario de la computadora puede especificar, con instrucciones programadas, que la operación aritmética se efectúe directamente con números decimales, para no tener que convertirlos a binarios.

Otros códigos decimales

Los códigos binarios para dígitos decimales requieren por lo menos cuatro bits por dígito. Es posible formular muchos códigos distintos acomodando cuatro bits en 10 combinaciones distintas. En la tabla 1-5 se muestran el código BCD y otros tres códigos representativos. Cada código utiliza sólo 10 combinaciones de bits, de las 16 posibles combinaciones que pueden formarse con cuatro bits. Las seis combinaciones no utilizadas en cada caso carecen de significado y deben evitarse.

Los códigos BCD y 2421 son ejemplos de códigos ponderados. En un código ponderado, se asigna a cada posición de bit un factor de ponderación (o peso) de modo que cada dígito pueda evaluarse sumando los pesos de todos los unos de la combinación codificada. Los pesos del código BCD son 8, 4, 2 y 1, que corresponden a los valores de potencia de 2 de cada bit. Por ejemplo, la asignación de bits 0110 se interpreta por los pesos como 6 decimal, porque $8 \times 0 + 4 \times 1 + 2 \times 1 + 1 \times 0 = 6$. La combinación de bits 1101, ponderada con los pesos respectivos 2421, da el equivalente decimal de $2 \times 1 + 4 \times 1 + 2 \times 0 + 1 \times 1 = 7$.

Tabla 1-5
Cuatro códigos binarios distintos para los dígitos decimales

Dígito decimal	BCD 8421	2421	Exceso-3	8 4-2-1
0	0000	0000	0011	0 0 0 0
1	0001	0001	0100	0 1 1 1
2	0010	0010	0101	0 1 1 0
3	0011	0011	0110	0 1 0 1
4	0100	0100	0111	0 1 0 0
5	0101	1011	1000	1 0 1 1
6	0110	1100	1001	1 0 1 0
7	0111	1101	1010	1 0 0 1
8	1000	1110	1011	1 0 0 0
9	1001	1111	1100	1 1 1 1
Combinaciones de bits no utilizadas	1010	0101	0000	0 0 0 1
	1011	0110	0001	0 0 1 0
	1100	0111	0010	0 0 1 1
	1101	1000	1101	1 1 0 0
	1110	1001	1110	1 1 0 1
	1111	1010	1111	1 1 1 0

Cabe señalar que es posible codificar algunos dígitos de dos formas en el código 2421. El 4 decimal se puede asignar a las combinaciones de bits 0100 o 1010, pues ambas dan un peso total de cuatro.

Los códigos 2421 y exceso-3 (excess-3) son ejemplos de códigos autocomplementadores. Tales códigos poseen la propiedad de que el complemento a nueve de un número decimal se obtiene directamente cambiando todos los ceros por unos y los unos por ceros en el código. Por ejemplo, el número decimal 395 se representa con 0110 1100 1000 en el código exceso-3. El complemento a nueve, 604, se representa con 1001 0011 0111, que se obtiene simplemente complementando cada bit del código (como se hace para obtener el complemento a uno de números binarios).

El código exceso-3 se usó en algunas computadoras viejas por su propiedad de autocomplementación. Se trata de un código no ponderado en el que cada combinación codificada se obtiene sumando 3 al valor binario correspondiente. Cabe señalar que el código BCD no se autocomplementa.

El código 8, 4, -2, -1 es un ejemplo de la asignación de pesos tanto positivos como negativos a un código decimal. En este caso, la combinación de bits 0110 se interpreta como un 2 decimal y se calcula a partir de $8 \times 0 + 4 \times 1 + (-2) \times 1 + (-1) \times 0 = 2$.

Código Gray

Los datos de salida de muchos sistemas físicos producen cantidades que son continuas. Es necesario convertir esos datos a una forma digital antes de aplicarse a un sistema digital. La información continua o analógica se convierte a una forma digital con un convertidor analógico a digital. Hay ocasiones en que conviene usar el código Gray que se muestra en la tabla 1-6 para representar los datos digitales obtenidos por conversión de datos analógicos. La ventaja del

Tabla 1-6
Código Gray

Código Gray	Equivalente decimal
0000	0
0001	1
0011	2
0010	3
0110	4
0111	5
0101	6
0100	7
1100	8
1101	9
1111	10
1110	11
1010	12
1011	13
1001	14
1000	15

código Gray sobre la sucesión continua de números binarios es que la diferencia entre dos números consecutivos cualesquiera en código Gray es de un solo bit. Por ejemplo, al pasar del 7 al 8, el código Gray cambia de 0100 a 1100. Sólo el primer bit cambia de 0 a 1; los otros tres bits no cambian. En cambio, en los números binarios, el cambio de 7 a 8 es de 0111 a 1000, o sea que los cuatro bits cambian de valor.

El código Gray se emplea en aplicaciones en las que la sucesión normal de números binarios podría generar un error o ambigüedad durante la transición de un número al siguiente. Si se usan números binarios, un cambio de 0111 a 1000 podría producir un número intermedio erróneo, como 1001, si el bit de la extrema derecha tarda más en cambiar su valor que los otros tres bits. El código Gray elimina este problema porque sólo un bit cambia de valor durante cualquier transición entre dos números.

Una aplicación típica del código Gray se da cuando datos analógicos se representan mediante un cambio continuo en la posición de un eje o flecha. El eje se divide en segmentos, y se asigna un número a cada segmento. Si hacemos que segmentos adyacentes correspondan a la sucesión del código Gray, se elimina la ambigüedad cuando la detección se efectúa en la línea que separa dos segmentos cualesquiera.

Código de caracteres ASCII

Muchas aplicaciones de las computadoras digitales requieren manipular datos que no sólo son números, sino también letras. Por ejemplo, una compañía de seguros con miles de asegurados utilizará una computadora para procesar sus archivos. Para representar los nombres y demás información pertinente, es necesario formular un código binario para las letras del alfabeto. Además, el mismo código binario deberá representar números y caracteres especiales (como $). Un conjunto de caracteres alfanuméricos es un conjunto de elementos que incluye los 10 dígitos decimales, las 26 letras del alfabeto y varios caracteres especiales. Un conjunto así contiene entre 36 y 64 elementos si sólo se incluyen letras mayúsculas, o entre 64 y 128 elementos si se incluyen mayúsculas y minúsculas. En el primer caso, se necesita un código binario de seis bits; en el segundo, se requiere uno de siete bits.

El código binario estándar para los caracteres alfanuméricos es ASCII (*American Standard Code for Information Interchange*, Código estándar americano para intercambio de información). Utiliza siete bits para codificar 128 caracteres, como se muestra en la tabla 1-7. Los siete bits del código se designan con b_1 a b_7, siendo b_7 el bit más significativo. La letra *A*, por ejemplo, se representa en ASCII como 1000001 (columna 100, fila 0001). El código ASCII contiene 94 caracteres gráficos que pueden imprimirse y 34 caracteres no imprimibles que se utilizan para diversas funciones de control. Los caracteres gráficos consisten en 26 letras mayúsculas (A a Z), 26 letras minúsculas (a a z), los 10 números (0 a 9) y 32 caracteres especiales imprimibles, como %, * y $.

Los 34 caracteres de control se designan con nombres abreviados en la tabla ASCII. Abajo de la tabla se presentan otra vez junto con el nombre de la función que desempeñan. Los caracteres de control sirven para determinar la ruta de los datos y organizar el texto impreso en un formato predefinido. Hay tres tipos de caracteres de control: creadores de formato, separadores de información y caracteres que controlan la comunicación. Los creadores de formato controlan la forma de imprimir e incluyen los controles ya conocidos de las máquinas de escribir, como retroceso (BS), tabulación horizontal (HT) y retorno de carro (CR). Los separadores de información sirven para separar los datos en divisiones como párrafos y páginas. Incluyen caracteres como el separador de registros (RS) y el separador de archivos (FS). Los caracteres

Tabla 1-7
Código estándar americano para intercambio de información (ASCII)

	$b_7b_6b_5$							
$b_4b_3b_2b_1$	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	“	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	‘	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	−	=	M	]	m	}
1110	SO	RS	.	>	N	∧	n	~
1111	SI	US	/	?	O	−	o	DEL

Caracteres de control

NUL	Nulo	DLE	Escape de enlace de datos
SOH	Inicio de cabecera	DC1	Control de dispositivo 1
STX	Inicio de texto	DC2	Control de dispositivo 2
ETX	Fin de texto	DC3	Control de dispositivo 3
EOT	Fin de transmisión	DC4	Control de dispositivo 4
ENQ	Pregunta	NAK	Acuse negativo
ACK	Acuse	SYN	Inactividad sincrónica
BEL	Campana	ETB	Fin de bloque de transmisión
BS	Retroceso	CAN	Cancelar
HT	Tabulación horizontal	EM	Fin de medio
LF	Salto de línea	SUB	Sustituto
VT	Tabulación vertical	ESC	Escape
FF	Salto de página	FS	Separador de archivos
CR	Retorno de carro	GS	Separador de grupos
SO	Desplazamiento afuera	RS	Separador de registros
SI	Desplazamiento adentro	US	Separador de unidades
SP	Espacio	DEL	Borrar

de control de comunicación desempeñan su función durante la transmisión de texto entre terminales remotas. Como ejemplos podemos citar STX (inicio de texto) y ETX (fin de texto), que sirven para encuadrar un mensaje de texto transmitido por líneas telefónicas.

ASCII es un código de siete bits, pero casi todas las computadoras manipulan una cantidad de ocho bits como una unidad llamada *byte*. Por ello, lo más común es almacenar un carácter ASCII por byte. El bit extra a veces se utiliza para otros fines, dependiendo de la aplicación.

Por ejemplo, algunas impresoras reconocen caracteres ASCII de ocho bits que tienen 0 en el bit más significativo. Los otros 128 caracteres de ocho bits que tienen 1 en el bit más significativo se utilizan para otros símbolos, como el alfabeto griego o letras cursivas.

Códigos para detectar errores

Cuando se desea detectar errores en la comunicación y en el procesamiento de datos, a veces se añade un octavo bit al carácter ASCII para indicar su paridad. El *bit de paridad* es un bit adicional que se incluye en un mensaje de modo que el número total de unos sea par o impar. Considere los dos caracteres siguientes y su paridad impar y par:

	Con paridad par	**Con paridad impar**
ASCII A = 1000001	01000001	11000001
ASCII T = 1010100	11010100	01010100

En ambos casos, insertamos un bit extra en la posición extrema izquierda del código para producir un número par de unos en el carácter, si se está usando paridad par, o un número impar de unos en el carácter, si se utiliza paridad impar. En general, se adopta sólo una de las dos paridades, siendo más común la paridad par.

El bit de paridad ayuda a detectar errores durante la transmisión de información de un lugar a otro. Esto se efectúa generando, para cada carácter, un bit de paridad par en el extremo transmisor. Los caracteres de ocho bits que incluyen bits de paridad se transmiten a su destino. En el extremo receptor se verifica la paridad de cada carácter recibido. Si un carácter no tiene un número par de unos, quiere decir que por lo menos un bit cambió de valor durante la transmisión. Este método detecta un error, tres errores o cualquier combinación impar de errores en cada carácter transmitido. Si hay un número par de errores, no se detectan. Podrían requerirse códigos adicionales de detección de errores para detectar una combinación par de errores.

Lo que se haga después de detectar un error dependerá de la aplicación de que se trate. Una posibilidad es solicitar la retransmisión del mensaje bajo el supuesto de que el error fue aleatorio y no volverá a presentarse. De este modo, si el receptor detecta un error de paridad, devolverá el carácter ASCII de control NAK (acuse negativo) que, si se usa paridad par, consiste en los ocho bits 10010101. Si no detecta ningún error, el receptor devuelve el carácter de control ACK (acuse), 00000110. El extremo transmisor responderá a un NAK transmitiendo otra vez el mensaje hasta que se reciba la paridad correcta. Si, después de varios intentos, la transmisión sigue teniendo errores, podría enviarse un mensaje al operador, pidiéndole buscar desperfectos en la ruta de transmisión.

1-8 ALMACENAMIENTO BINARIO Y REGISTROS

La información binaria de una computadora digital debe existir físicamente en algún medio de almacenamiento capaz de guardar bits individuales. Una *celda binaria* es un dispositivo que tiene dos estados estables y puede almacenar un bit de información. La entrada de la celda recibe señales de excitación que colocan a la celda en uno de los dos estados. La salida de la celda es una cantidad física que distingue entre los dos estados. La información almacenada en una celda es 1 si la celda está en uno de sus estados estables, y 0 cuando está en el otro.

Registros

Un *registro* es un grupo de celdas binarias. Un registro con n celdas es capaz de almacenar cualquier cantidad discreta de información que contenga n bits. El estado de un registro es una n-tupla de unos y ceros, en la que cada bit designa el estado de una celda del registro. El contenido de un registro es función de la interpretación que se da a la información ahí almacenada. Considere, por ejemplo, un registro de 16 bits con el contenido siguiente:

$$1100001111001001$$

Un registro con 16 celdas puede estar en uno de 2^{16} posibles estados. Si suponemos que el contenido del registro representa un entero binario, el registro podrá almacenar cualquier número binario entre 0 y $2^{16} - 1$. En nuestro ejemplo, el contenido del registro es el equivalente binario del número decimal 50121. Si suponemos que el registro almacena los caracteres alfanuméricos de un código de ocho bits, el contenido del registro sería dos caracteres. En el caso del código ASCII con un bit de paridad par en la octava posición de bit más significativa, el registro contiene los dos caracteres C (ocho bits de la izquierda) e I (ocho bits de la derecha). Por otra parte, si interpretamos el contenido del registro como cuatro dígitos decimales representados en un código de cuatro bits, el contenido del registro será un número decimal de cuatro dígitos. En el código exceso-3, el registro contiene el número decimal 9096. El contenido del registro no tiene sentido en BCD porque la combinación de bits 1100 no tiene asignado ningún dígito decimal. Este ejemplo hace evidente que un registro puede almacenar elementos discretos de información y que la misma configuración de bits puede interpretarse de diferentes maneras, dependiendo del tipo de los datos.

Transferencia de registro

Un sistema digital se caracteriza por sus registros y los componentes que efectúan el procesamiento de datos. La operación de *transferencia de registro* es básica en los sistemas digitales. Consiste en una transferencia de información binaria de un conjunto de registros a otro. La transferencia podría ser directa de un registro a otro, o podría pasar por circuitos procesadores de datos para efectuar una operación. La figura 1-1 ilustra la transferencia de información entre registros y muestra gráficamente la transferencia de información de un teclado a un registro en la unidad de memoria. Se supone que la unidad de entrada tiene un teclado, un circuito de control y un registro de entrada. Cada vez que se pulsa una tecla, el control introduce un código equivalente de carácter alfanumérico, de ocho bits, en el registro de entrada. Supondremos que el código empleado es ASCII con bit de paridad impar. La información del registro de entrada se transfiere a las ocho celdas menos significativas de un registro del procesador. Después de cada transferencia, el registro de entrada se despeja para que el control pueda insertar un nuevo código de ocho bits cuando se vuelva a accionar el teclado. Cada carácter de ocho bits transferido al registro del procesador va precedido por un desplazamiento del carácter anterior a las ocho celdas que siguen a la izquierda. Una vez que se transfieren cuatro caracteres, el registro del procesador se llena y su contenido se transfiere a un registro de memoria. El contenido almacenado en el registro de memoria que se muestra en la figura 1-1 provino de la transferencia de los caracteres “J”, “O”, “H” y “N” después de pulsarse las cuatro teclas correspondientes.

Para procesar cantidades discretas de información en forma binaria, la computadora necesita dispositivos para retener los datos que se procesarán, así como elementos de circuito que

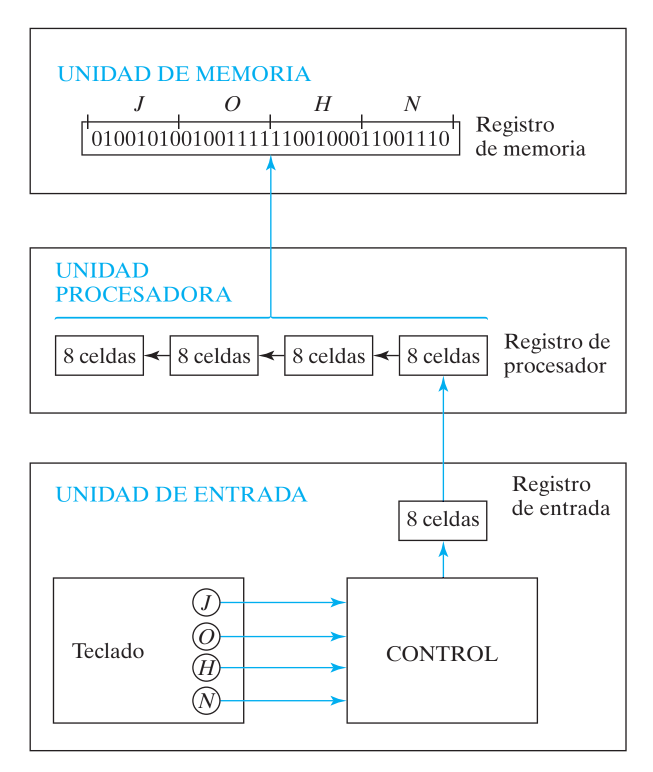


FIGURA 1-1
Transferencia de información con registros

manipulen bits individuales de información. El dispositivo que se utiliza con mayor frecuencia para retener datos es el registro. La manipulación de variables binarias se efectúa mediante circuitos lógicos digitales. La figura 1-2 ilustra el proceso de sumar dos números binarios de 10 bits. La unidad de memoria, que normalmente consta de millones de registros, aparece sólo con tres registros en el diagrama. La parte de la unidad procesadora que se muestra consiste en tres registros —*R*1, *R*2 y *R*3— además de circuitos lógicos digitales que manipulan los bits de *R*1 y *R*2 y transfieren a *R*3 un número binario igual a su suma aritmética. Los registros de memoria almacenan información y no pueden procesar los dos operandos, pero es posible transferir la información que contienen a registros del procesador. Los resultados obtenidos en los registros del procesador se pueden transferir de vuelta a un registro de la memoria para almacenarse hasta que se necesiten otra vez. En el diagrama, el contenido de dos operandos se transfiere de dos registros de memoria a *R*1 y *R*2. Los circuitos de lógica digital producen la suma, que se transfiere al registro *R*3. El contenido de *R*3 se puede transferir entonces a uno de los registros de memoria.

Los últimos dos ejemplos ilustran de forma muy sencilla las capacidades de flujo de información de un sistema digital. Los registros del sistema son los elementos básicos para almacenar y retener la información binaria. Los circuitos de lógica digital procesan la información binaria almacenada en los registros. Los circuitos de lógica digital y los registros se estudiarán en los capítulos 2 a 6. La unidad de memoria se explica en el capítulo 7. El nivel de transferencia de registros para describir y diseñar sistemas digitales se trata en el capítulo 8.

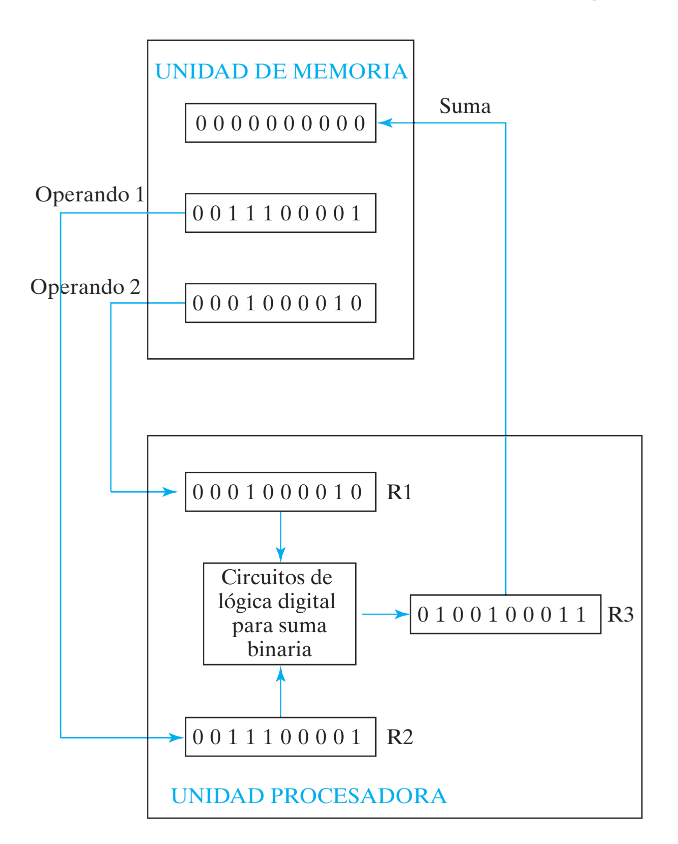


FIGURA 1-2
Ejemplo de procesamiento de información binaria

1-9 LÓGICA BINARIA

La lógica binaria se ocupa de variables que adoptan dos valores discretos y de operaciones que asumen un significado lógico. Los dos valores que pueden adoptar las variables reciben diferentes nombres (*verdadero* y *falso*, *sí* y *no*, etcétera), pero para nuestros fines es conveniente pensar en ellos en términos de bits y asignarles los valores 1 y 0. La lógica binaria que se presenta en esta sección equivale a un álgebra llamada álgebra booleana, que se estudiará formalmente en el capítulo 2. El propósito de esta sección es hacer una introducción heurística al álgebra booleana y relacionarla con los circuitos de lógica digital y las señales binarias.

Definición de lógica binaria

La lógica binaria consiste en variables binarias y operaciones lógicas. Las variables se designan con letras del alfabeto, como A, B, C, x, y, z, etcétera, y cada variable tiene dos y sólo dos posibles valores: 1 y 0. Hay tres operaciones lógicas básicas: AND, OR y NOT.

1. AND: Esta operación se representa con un punto u omitiendo el operador. Por ejemplo, $x \cdot y = z$ o $xy = z$ se lee "x AND y es igual a z". La operación lógica AND significa que $z = 1$ si y sólo si $x = 1$ y $y = 1$; de lo contrario, $z = 0$. (Recordemos que x, y y z son variables binarias y sólo pueden valer 1 o 0.)
2. OR: Esta operación se representa con un signo más. Por ejemplo, $x + y = z$ se lee "x OR y es igual a z", y significa que $z = 1$ si $x = 1$ *o* si $y = 1$ *o* si $x = 1$ y $y = 1$. Si $x = 0$ y $y = 0$, entonces $z = 0$.
3. NOT: Esta operación se representa con un apóstrofo (y a veces con una testa). Por ejemplo, $x' = z$ (o $\bar{x} = z$) se lee como "no x es igual a z" y significa que z es lo contrario de x. Dicho de otro modo, si $x = 1$, entonces $z = 0$; pero si $x = 0$, entonces $z = 1$. La operación NOT también se llama operación de complemento, ya que cambia un 1 por 0 y un 0 por 1.

La lógica binaria se parece a la aritmética binaria, y las operaciones AND y OR tienen similitudes con la multiplicación y la suma, respectivamente. De hecho, los símbolos que se usan para AND y OR son los mismos que los empleados para la multiplicación y la suma. No obstante, la lógica binaria no debe confundirse con la aritmética binaria. Debemos tener presente que una variable aritmética designa a un número que podría consistir en muchos dígitos. Una variable lógica siempre es 0 o 1. Por ejemplo, en aritmética binaria tenemos $1 + 1 = 10$ (léase: "uno más uno es igual a dos"), mientras que en lógica binaria tenemos $1 + 1 = 1$ (léase: "uno OR uno es igual a uno").

Para cada combinación de los valores de x y y, la definición de la operación lógica especifica un valor de z. Dichas definiciones se pueden presentar en forma compacta con *tablas de verdad*. Una tabla de verdad es una tabla de todas las posibles combinaciones de las variables, y muestra la relación entre los valores que las variables pueden adoptar y el resultado de la operación. Las tablas de verdad de las operaciones AND y OR con variables x y y se obtienen enumerando todos los posibles valores que pueden tener las variables cuando se les combina en pares. Luego, se anota en una fila aparte el resultado de la operación para cada combinación. En la tabla 1-8 se presentan las tablas de verdad para AND, OR y NOT. Estas tablas muestran claramente la definición de las operaciones.

Compuertas lógicas

Las compuertas lógicas son circuitos electrónicos que operan con una o más señales de entrada para producir una señal de salida. En los sistemas digitales, las señales eléctricas, que po-

Tabla 1-8
Tablas de verdad de operaciones lógicas

AND			OR			NOT	
x	y	$x \cdot y$	x	y	$x + y$	x	x'
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

drían ser voltajes o corrientes, existen con uno de dos valores reconocibles. Los circuitos operados por voltaje responden a dos niveles de voltaje distintos que representan una variable binaria cuyo valor es 1 lógico o 0 lógico. Por ejemplo, un sistema digital dado podría definir el 0 lógico como una señal de 0 volts, y el 1 lógico, como una señal de 4 volts. En la práctica, cada nivel de voltaje tiene un intervalo aceptable, como se indica en la figura 1-3. Las terminales de entrada de los circuitos digitales aceptan señales binarias dentro del intervalo permisible y responden en las terminales de salida con señales binarias que están dentro del intervalo especificado. La región intermedia entre las regiones permitidas sólo se cruza durante las transiciones de estado. Cualquier información deseada para computación o control se puede manipular haciendo pasar señales binarias por diversas combinaciones de compuertas lógicas, y cada señal representa una variable binaria dada.

En la figua 1-4 se incluyen los símbolos gráficos con que se representan los tres tipos de compuertas. Las compuertas son bloques de hardware que producen señales de salida equivalentes al 1 lógico o al 0 lógico cuando se satisfacen los requisitos lógicos de entrada. Las señales de entrada x y y de las compuertas lógicas AND y OR podrían existir en uno de cuatro posibles estados: 00, 10, 11 o 01. Estas señales de entrada se muestran en la figura 1-5 junto con

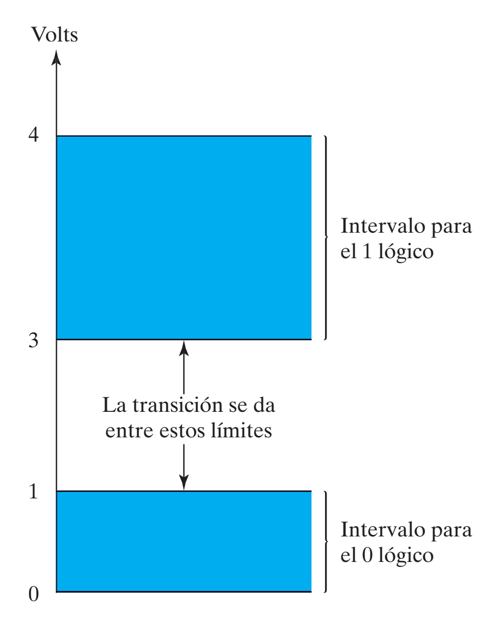


FIGURA 1-3
Ejemplo de señales binarias

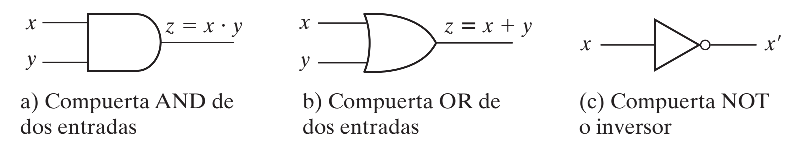


FIGURA 1-4
Símbolos para los circuitos lógicos digitales

x 0 1 1 0 0

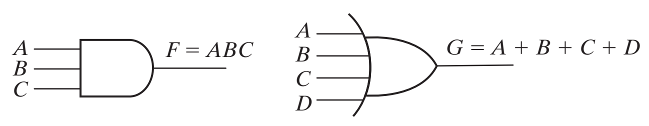
y 0 0 1 1 0

AND: $x \cdot y$ 0 0 1 0 0

OR: $x + y$ 0 1 1 1 0

NOT: x' 1 0 0 1 1

FIGURA 1-5
Señales de entrada-salida de compuertas



a) Compuerta AND de tres entradas

b) Compuerta OR de cuatro entradas

FIGURA 1-6
Compuertas con múltiples entradas

la señal de salida correspondiente a cada compuerta. Los diagramas de temporización ilustran la respuesta de cada compuerta a las cuatro combinaciones de señales de entrada. El eje horizontal del diagrama de temporización representa tiempo, mientras que el eje vertical muestra cómo cambia la señal entre los dos posibles niveles de voltaje. El nivel bajo representa el 0 lógico y el nivel alto representa el 1 lógico. La compuerta AND responde con una señal de salida de 1 lógico cuando ambas señales de entrada son 1 lógico. La compuerta OR responde con una señal de salida de 1 lógico cuando cualquier señal de entrada es 1 lógico. La compuerta NOT se conoce comúnmente como inversor, y en el diagrama de temporización es evidente el porqué: la señal de salida invierte el sentido lógico de la señal de entrada.

Las compuertas AND y OR pueden tener más de dos entradas. En la figura 1-6 se representa una compuerta AND con tres entradas y una compuerta OR con cuatro entradas. La compuerta AND de tres entradas responde con una salida de 1 lógico si las tres entradas son 1 lógico, y con 0 lógico si cualquiera de las entradas es 0 lógico. La compuerta OR de cuatro entradas responde con 1 lógico si cualquier entrada es 1 lógico; su salida sólo será 0 lógico si todas sus entradas son 0 lógico.

PROBLEMAS

1-1 Enumere los números octales y hexadecimales del 16 al 32. Utilizando A y B como últimos dos dígitos, enumere los números del 10 al 26 en base 12.

1-2 ¿Cuántos bytes hay exactamente en un sistema que contiene a) 32K bytes, b) 64M bytes, y c) 6.4G bytes?

1-3 Dé el número binario más grande que se puede expresar con 12 bits. Dé su equivalente decimal y hexadecimal.

1-4 Convierta a decimal los números que siguen en las bases indicadas: $(4310)_5$ y $(198)_{12}$.

1-5 Determine en cada caso la base de los números, de modo que las operaciones sean correctas: a) 14/2 = 5; b) 54/4 = 13, y c) 24 + 17 = 40.

1-6 La solución de la ecuación cuadrática $x^2 - 11x + 22 = 0$ es $x = 3$ y $x = 6$. ¿Qué base tienen los números?

1-7 Exprese estos números en decimal: $(10110.0101)_2$, $(16.5)_{16}$ y $(26.24)_8$.

1-8 Convierta estos números binarios a hexadecimal y decimal: a) 1.11010, b) 1110.10. Explique por qué la respuesta decimal a b) es 8 veces la de a).

1-9 Convierta el número hexadecimal 68BE a binario y, de binario, conviértalo a octal.

1-10 Convierta el número decimal 345 a binario de dos maneras: a) conviértalo directamente a binario; b) conviértalo primero a hexadecimal, y luego de hexadecimal a binario. ¿Qué método es más rápido?

1-11 Resuelva los siguientes problemas de conversión:
a) Convierta el número decimal 34.4375 a binario.
b) Calcule el equivalente binario de 1/3 hasta ocho posiciones. Luego conviértalo de binario a decimal. ¿Qué tan cercano a 1/3 es el resultado?
c) Convierta el resultado binario de b) a hexadecimal. Luego convierta el resultado a decimal. ¿La respuesta es la misma?

1-12 Sume y multiplique los números siguientes sin convertirlos a decimal.
a) Números binarios 1011 y 101.
b) Números hexadecimales 2E y 34.

1-13 Realice esta división en binario: 1011111 ÷ 101.

1-14 Obtenga el complemento a nueve y a diez de los números decimales siguientes:
a) 98127634 b) 72049900 c) 10000000 d) 00000000.

1-15 a) Obtenga el complemento a 16 de AF3B.
b) Convierta AF3B a binario.
c) Obtenga el complemento a dos del resultado de b).
d) Convierta la respuesta de c) a hexadecimal y compárela con la respuesta de a).

1-16 Obtenga los complementos a uno y a dos de estos números binarios:
a) 11101010 b) 01111110 c) 00000001 d) 10000000 (e) 00000000.

1-17 Efectúe la resta de los siguientes números sin signo utilizando el complemento a 10 del sustraendo. Si el resultado es negativo, obtenga su complemento a 10 y antepóngale un signo menos. Compruebe sus respuestas.
a) 7188 − 3049 b) 150 − 2100 c) 2997 − 7992 d) 1321 − 375

1-18 Efectúe la resta de los siguientes números binarios sin signo utilizando el complemento a dos del sustraendo. Si el resultado es negativo, obtenga su complemento a dos y antepóngale un signo menos.
a) 11011 − 11001 b) 110100 − 10101 c) 1011 − 110000 d) 101010 − 101011

1-19 Los números decimales que siguen se presentan en forma de magnitud con signo: +9826 y +801. Conviértalos a la forma de complemento a 10 con signo y realice las operaciones siguientes (tome nota de que la suma es +10627 y requiere seis dígitos):
a) (+9826) + (+801) b) (+9826) + (−801)
c) (−9826) + (+801) d) (−9826) + (−801)

1-20 Convierta los números decimales +61 y +27 a binario empleando la representación de complemento a dos con signo y suficientes dígitos para dar cabida a los números. Luego efectúe el equivalente binario de (+27) + (−61), (−27) + (+61) y (−27) + (−61). Convierta las respuestas a decimal y verifique que sean correctas.

1-21 Convierta el número decimal 9126 a los códigos BCD y ASCII. En el caso de ASCII, añada un bit de paridad impar a la izquierda.

1-22 Represente los números decimales sin signo 965 y 672 en BCD y luego muestre los pasos necesarios para obtener su suma.

1-23 Formule un código binario ponderado para los dígitos decimales empleando los pesos 6, 3, 1, 1.

1-24 Represente el número decimal 6027 en a) BCD, b) código exceso-3, y c) código 2421.

1-25 Obtenga el complemento a nueve de 6027 y expréselo en código 2421. Demuestre que el resultado es el complemento a uno de la respuesta al inciso c) del problema 1-24. Esto demuestra que el código 2421 se autocomplementa.

1-26 Asigne un código binario ordenado a los 52 naipes de la baraja. Utilice el número mínimo de bits.

1-27 Escriba la expresión "G. Boole" en ASCII empleando un código de ocho bits. Incluya el punto y el espacio. Trate el bit de extrema izquierda de cada carácter como bit de paridad. Cada código de 8 bits deberá tener paridad par. (George Boole fue un matemático del siglo XIX. El álgebra Booleana, que se estudiará en el capítulo siguiente, lleva su nombre.)

1-28 Decodifique el código ASCII siguiente: 1001010 1100001 1101110 1100101 0100000 1000100 1101111 1100101.

1-29 La que sigue es una cadena de caracteres ASCII cuyos patrones de bits se han convertido a hexadecimal para que no ocupen tanto espacio: 4A EF 68 6E 20 C4 EF E5. De los ocho bits de cada par de dígitos, el de la extrema izquierda es un bit de paridad. Los bits restantes son el código ASCII.

a) Conviértalos a bits y decodifique el ASCII.
b) Determine la paridad empleada: impar o par.

1-30 ¿Cuántos caracteres imprimibles hay en ASCII? ¿Cuántos de ellos son caracteres especiales (ni letras ni números)?

1-31 ¿Qué bit es preciso complementar para cambiar una letra ASCII de mayúscula a minúscula, y viceversa?

1-32 El estado de un registro de 12 bits es 100010010111. ¿Qué contiene si representa

a) tres dígitos decimales en BCD?
b) tres dígitos decimales en código exceso-3?
c) tres dígitos decimales en código 84-2-1?
d) un número binario?

1-33 Haga una lista con el código ASCII de los 10 dígitos decimales, con un bit de paridad par en la posición de extrema izquierda.

1-34 Suponga una compuerta AND de tres entradas cuya salida es F y una compuerta OR de tres entradas cuya salida es G. Las entradas son A, B y C. Muestre las señales (en un diagrama de temporización similar al de la figura 1-5) de las salidas F y G en función de las tres entradas ABC. Utilice las ocho posibles combinaciones de ABC.

REFERENCIAS

1. Cavanagh, J. J. 1984. *Digital Computer Arithmetic*. Nueva York: McGraw-Hill.
2. Schmid, H. 1974. *Decimal Computation*. Nueva York: John Wiley.
3. Mano, M. M. 1988. *Computer Engineering: Hardware Design*. Englewood Cliffs, NJ: Prentice-Hall.
4. Nelson, V. P., H. T. Nagle, J. D. Irwin y B. D. Carroll 1997. *Digital Logic Circuit Analysis and Design*. Upper Saddle River, NJ: Prentice-Hall.

2 Álgebra booleana y compuertas lógicas

2-1 DEFINICIONES BÁSICAS

El álgebra booleana, al igual que todos los sistemas matemáticos deductivos, se define con un conjunto de elementos, un conjunto de operadores y varios axiomas o postulados no demostrados. Un *conjunto* de elementos es cualquier colección de objetos con alguna propiedad en común. Si S es un conjunto y x y y son ciertos objetos, entonces $x \in S$ denota que x es un miembro del conjunto S, y $y \notin S$ denota que y no es un elemento de S. Un conjunto que tiene un número enumerable de elementos se especifica con llaves: $A = \{1, 2, 3, 4\}$, es decir, los elementos de A son los números 1, 2, 3 y 4. Un *operador binario* definido sobre un conjunto S de elementos es una regla que asigna a cada par de elementos de S un elemento único de S. Por ejemplo, consideremos la relación $a * b = c$. Decimos que $*$ es un operador binario si especifica una regla para encontrar c a partir del par (a, b) y si además $a, b, c \in S$. Sin embargo, $*$ no es un operador binario si $a, b \in S$ y la regla encuentra que $c \notin S$.

Los postulados de un sistema matemático constituyen los supuestos básicos a partir de los cuales es posible deducir las reglas, teoremas y propiedades del sistema. Los postulados más comunes que se utilizan para formular diversas estructuras algebraicas son:

1. *Cerradura*. Un conjunto S es cerrado respecto a un operador binario si, por cada par de elementos de S, el operador especifica una regla para obtener un elemento único de S. Por ejemplo, el conjunto de los números naturales $N = \{1, 2, 3, 4, \ldots\}$ es cerrado respecto al operador binario más $(+)$ por las reglas de la suma aritmética, ya que, para cualquier $a, b \in N$, obtenemos un $c \in N$ único por la operación $a + b = c$. El conjunto de los números naturales no es cerrado respecto al operador binario menos $(-)$ por las reglas de la resta aritmética, porque $2 - 3 = -1$ y $2, 3 \in N$ pero $(-1) \notin N$.
2. *Ley asociativa*. Decimos que un operador binario $*$ sobre un conjunto S es asociativo si

$$(x * y) * z = x * (y * z) \text{ para todos } x, y, z \in S$$

3. *Ley conmutativa*. Decimos que un operador binario $*$ sobre un conjunto S es conmutativo si

$$x * y = y * x \text{ para todos } x, y \in S$$

4. *Elemento de identidad*. Decimos que un conjunto S tiene un elemento de identidad respecto a una operación binaria $*$ sobre S si existe un elemento $e \in S$ con la propiedad

$$e * x = x * e = x \text{ para todos } x \in S$$

Ejemplo: El elemento 0 es un elemento de identidad respecto a la operación $+$ sobre el conjunto de los enteros $I = \{..., -3, -2, -1, 0, 1, 2, 3, \ldots\}$, porque

$$x + 0 = 0 + x = x \text{ para cualquier } x \in I$$

El conjunto de los números naturales, N, no tiene elemento de identidad porque 0 no pertenece al conjunto.

5. *Inverso.* Decimos que un conjunto S, que tiene el elemento de identidad e respecto a un operador $*$, tiene un inverso si, para todo $x \in S$, existe un elemento $y \in S$ tal que

$$x * y = e$$

Ejemplo: En el conjunto de enteros, I, donde $e = 0$, el inverso de un elemento a es $(-a)$, ya que $a + (-a) = 0$.

6. *Ley distributiva*. Si $*$ y $\cdot$ son dos operadores binarios sobre un conjunto S, decimos que $*$ es distributivo sobre $\cdot$ si

$$x * (y \cdot z) = (x * y) \cdot (x * z)$$

Un ejemplo de estructura algebraica es un *campo*. Un campo es un conjunto de elementos, junto con dos operadores binarios, cada uno de los cuales posee las propiedades 1 a 5 y, combinados, la propiedad 6. El conjunto de los números reales, junto con los operadores binarios $+$ y $\cdot$, forman el campo de los números reales. Este campo es la base de la aritmética y el álgebra ordinaria. Los operadores y postulados significan lo siguiente:

El operador binario $+$ define la suma.

La identidad aditiva es 0.

El inverso aditivo define la resta.

El operador binario $\cdot$ define la multiplicación.

La identidad multiplicativa es 1.

El inverso multiplicativo de $a = 1/a$ define la división, es decir, $a \cdot 1/a = 1$.

La única ley distributiva válida es la de $\cdot$ sobre $+$:

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$$

2-2 DEFINICIÓN AXIOMÁTICA DEL ÁLGEBRA BOOLEANA

En 1854, George Boole introdujo un tratamiento sistemático de la lógica y desarrolló, con este fin, un sistema algebraico que ahora llamamos *álgebra booleana*. En 1938, C. E. Shannon introdujo un álgebra booleana de dos valores llamada *álgebra de conmutación* y demostró que

las propiedades de los circuitos eléctricos de conmutación biestables podían representarse con esa álgebra. Para definir formalmente el álgebra booleana, utilizaremos los postulados formulados por E. V. Huntington en 1904.

El álgebra booleana es una estructura algebraica definida por un conjunto de elementos, B, junto con dos operadores binarios, $+$ y $\cdot$, a condición de que se satisfagan los postulados siguientes (de Huntington):

1. a) Cerradura respecto al operador $+$.
b) Cerradura respecto al operador $\cdot$.

2. a) Un elemento de identidad con respecto a $+$, designado por 0: $x + 0 = 0 + x = x$.
b) Un elemento de identidad con respecto a $\cdot$, designado por 1: $x \cdot 1 = 1 \cdot x = x$.

3. a) Conmutativa respecto a $+$: $x + y = y + x$.
b) Conmutativa respecto a $\cdot$: $x \cdot y = y \cdot x$.

4. a) $\cdot$ es distributivo sobre $+$: $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$.
b) $+$ es distributivo sobre $\cdot$: $x + (y \cdot z) = (x + y) \cdot (x + z)$.

5. Para cada elemento $x \in B$, existe un elemento $x' \in B$ (llamado complemento de x) tal que a) $x + x' = 1$ y b) $x \cdot x' = 0$.

6. Existen por lo menos dos elementos $x, y \in B$ tales que $x \neq y$.

Al comparar el álgebra booleana con la aritmética y el álgebra ordinaria (el campo de los números reales), observamos las siguientes diferencias:

1. Los postulados de Huntington no incluyen la ley asociativa. Sin embargo, esta ley se cumple para el álgebra booleana y se puede derivar (para ambos operadores) de los otros postulados.
2. La ley distributiva de $+$ sobre $\cdot$, es decir, $x + (y \cdot z) = (x + y) \cdot (x + z)$, es válida para el álgebra booleana, pero no para el álgebra ordinaria.
3. El álgebra booleana no tiene inversos aditivos ni multiplicativos; por tanto, no hay operaciones de resta ni de división.
4. El postulado 5 define un operador llamado *complemento* que no existe en el álgebra ordinaria.
5. El álgebra ordinaria se ocupa de los números reales, que constituyen un conjunto infinito de elementos. El álgebra booleana se ocupa de un conjunto de elementos B que todavía no hemos definido, pero en el álgebra booleana de dos valores que definiremos a continuación (y que nos interesará para nuestro uso subsecuente de esta álgebra), B se define como un conjunto con sólo dos elementos, 0 y 1.

El álgebra booleana se parece al álgebra ordinaria en algunos sentidos. La selección de los símbolos $+$ y $\cdot$ es intencional para que quienes ya conocen el álgebra ordinaria puedan efectuar con más facilidad manipulaciones algebraicas booleanas. Aunque podemos aprovechar algunos conocimientos del álgebra ordinaria para trabajar con el álgebra booleana, el principiante debe tener cuidado de no usar las reglas del álgebra ordinaria en casos en que no son válidas.

Es importante distinguir entre los elementos del conjunto de una estructura algebraica y las variables de un sistema algebraico. Por ejemplo, los elementos del campo de los números reales son números, mientras que variables como a, b, c, etcétera, que se usan en el álgebra ordinaria, son símbolos que representan números reales. Asimismo, en el álgebra booleana, definimos los elementos del conjunto B, y las variables como x, y y z son meramente símbolos que representan a los elementos. A estas alturas, es importante entender que, para tener un álgebra booleana, es preciso mostrar:

1. los elementos del conjunto B,
2. las reglas de operación de los dos operadores binarios, y
3. que el conjunto de elementos, B, junto con los dos operadores, satisface los seis postulados de Huntington.

Podemos formular muchas álgebras booleanas, dependiendo de los elementos que escojamos para B y de las reglas de operación. De aquí en adelante, nos ocuparemos únicamente de un álgebra booleana de dos valores, es decir, una que sólo tiene dos elementos. El álgebra booleana de dos valores tiene aplicaciones en teoría de conjuntos (el álgebra de clases) y en la lógica proposicional. Lo que nos interesa aquí es la aplicación del álgebra booleana a los circuitos tipo compuerta.

Álgebra booleana de dos valores

Un álgebra booleana de dos valores se define sobre un conjunto de dos elementos, $B = \{1, 0\}$, con las reglas para los dos operadores binarios, $+$ y $\cdot$, que se muestran en las siguientes tablas de operador (la regla del operador de complemento es para verificar el postulado 5):

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

x	x'
0	1
1	0

Estas reglas son exactamente las mismas que las de las operaciones AND, OR y NOT, respectivamente, definidas en la tabla 1-8. Ahora debemos demostrar que los postulados de Huntington son válidos para el conjunto $B = \{0, 1\}$ y los dos operadores binarios que hemos definido.

1. La *cerradura* es obvia por las tablas, pues el resultado de todas las operaciones es 1 o bien 0, y $1, 0 \in B$.

2. En las tablas vemos que

   a) $0 + 0 = 0$ $\quad 0 + 1 = 1 + 0 = 1;$

   b) $1 \cdot 1 = 1$ $\quad 1 \cdot 0 = 0 \cdot 1 = 0.$

   Esto establece los dos *elementos de identidad*, 0 para $+$ y 1 para $\cdot$, definidos por el postulado 2.

3. Las leyes *conmutativas* son obvias por la simetría de las tablas de los operadores binarios.

4. **a)** Es posible demostrar que la ley *distributiva* $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$ se cumple preparando una tabla de verdad con todos los posibles valores de x, y y z. Para cada combinación, deducimos $x \cdot (y + z)$ y demostramos que su valor es igual al de $(x \cdot y) + (x \cdot z)$.

x	y	z	$y + z$	$x \cdot (y + z)$	$x \cdot y$	$x \cdot z$	$(x \cdot y) + (x \cdot z)$
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

b) Podemos demostrar que se cumple la ley *distributiva* de $+$ sobre $\cdot$ preparando una tabla de verdad similar a la anterior.

5. La tabla del complemento permite demostrar fácilmente que

a) $x + x' = 1$, ya que $0 + 0' = 0 + 1 = 1$ y $1 + 1' = 1 + 0 = 1$.

b) $x \cdot x' = 0$, ya que $0 \cdot 0' = 0 \cdot 1 = 0$ y $1 \cdot 1' = 1 \cdot 0 = 0$, lo que verifica el postulado 5.

6. El postulado 6 se satisface porque el álgebra booleana de dos valores tiene dos elementos, 1 y 0, y $1 \neq 0$.

Acabamos de establecer un álgebra booleana de dos valores que tiene un conjunto de dos elementos, 1 y 0, dos operadores binarios con reglas de operación equivalentes a las operaciones AND y OR, y un operador de complemento equivalente al operador NOT. Así pues, el álgebra booleana ha quedado definida de manera matemática formal y se ha demostrado que es equivalente a la lógica binaria que presentamos heurísticamente en la sección 1-9. La presentación heurística ayuda a entender la aplicación del álgebra booleana a los circuitos tipo compuerta. La presentación formal es necesaria para desarrollar los teoremas y propiedades del sistema algebraico. Los ingenieros también llaman "álgebra de conmutación" al álgebra booleana de dos valores que definimos en esta sección. A fin de hacer hincapié en las similitudes entre el álgebra booleana de dos valores y otros sistemas binarios, llamamos "lógica binaria" a esta álgebra en la sección 1-9. De aquí en adelante, omitiremos el calificativo "de dos valores" al hablar de álgebra booleana en las explicaciones.

2-3 TEOREMAS Y PROPIEDADES BÁSICOS DEL ÁLGEBRA BOOLEANA

Dualidad

Se han presentado los postulados de Huntington por pares y se han designado como parte a) y parte b). Es posible obtener una parte de la otra si se intercambian los operadores binarios y los elementos de identidad. Esta importante propiedad del álgebra booleana se denomina *principio de dualidad*, y establece que toda expresión algebraica que pueda deducirse de los postulados del álgebra booleana seguirá siendo válida si se intercambian los operadores y los elementos de identidad. En un álgebra booleana de dos valores, los elementos de identidad y

los elementos del conjunto, B, son los mismos: 1 y 0. El principio de dualidad tiene muchas aplicaciones. Si queremos el *dual* de una expresión algebraica, simplemente intercambiamos los operadores OR y AND y sustituimos los unos por ceros y los ceros por unos.

Teoremas básicos

En la tabla 2-1 se presentan seis teoremas del álgebra booleana y cuatro de sus postulados. La notación se simplifica omitiendo el operador binario $\cdot$ en los casos en que ello no causa confusión. Los teoremas y postulados que se presentan son las relaciones más básicas del álgebra booleana. Los teoremas, al igual que los postulados, se presentan por pares; cada relación es el dual de su pareja. Los postulados son axiomas básicos de la estructura algebraica y no requieren demostración. Los teoremas deben demostrarse a partir de los postulados. A continuación se presentan las demostraciones de los teoremas con una variable. A la derecha se indica el número del postulado que justifica cada paso de la demostración.

TEOREMA 1a): $x + x = x$.

$x + x = (x + x) \cdot 1$	por el postulado:	2b)
$= (x + x)(x + x')$		5a)
$= x + xx'$		4b)
$= x + 0$		5b)
$= x$		2a)

TEOREMA 1b): $x \cdot x = x$.

$x \cdot x = xx + 0$	por el postulado:	2a)
$= xx + xx'$		5b)
$= x(x + x')$		4a)
$= x \cdot 1$		5a)
$= x$		2b)

Tabla 2-1
Postulados y teoremas del álgebra booleana

Postulado 2	a)	$x + 0 = x$	b)	$x \cdot 1 = x$
Postulado 5	a)	$x + x' = 1$	b)	$x \cdot x' = 0$
Teorema 1	a)	$x + x = x$	b)	$x \cdot x = x$
Teorema 2	a)	$x + 1 = 1$	b)	$x \cdot 0 = 0$
Teorema 3, involución		$(x')' = x$		
Postulado 3, conmutatividad	a)	$x + y = y + x$	b)	$xy = yx$
Teorema 4, asociatividad	a)	$x + (y + z) = (x + y) + z$	b)	$x(yz) = (xy)z$
Postulado 4, distributividad	a)	$x(y + z) = xy + xz$	b)	$x + yz = (x + y)(x + z)$
Teorema 5, DeMorgan	a)	$(x + y)' = x'y'$	b)	$(xy)' = x' + y'$
Teorema 6, absorción	a)	$x + xy = x$	b)	$x(x + y) = x$

Observe que el teorema 1b) es el dual del teorema 1a) y que cada uno de los pasos de la demostración en la parte b) es el dual de la parte a). Cualquier teorema dual se puede deducir de forma similar, partiendo de la demostración de su par correspondiente.

TEOREMA 2a): $x + 1 = 1$.

$x + 1 = 1 \cdot (x + 1)$	por el postulado:	2b)
$= (x + x')(x + 1)$		5a)
$= x + x' \cdot 1$		4b)
$= x + x'$		2b)
$= 1$		5a)

TEOREMA 2b): $x \cdot 0 = 0$ por dualidad.

TEOREMA 3: $(x')' = x$. Del postulado 5, tenemos $x + x' = 1$ y $x \cdot x' = 0$, lo que define al complemento de x. El complemento de x' es x y también es $(x')'$. Por tanto, dado que el complemento es único, tenemos que $(x')' = x$.

Los teoremas en los que intervienen dos o tres variables se pueden demostrar algebraicamente a partir de los postulados y los teoremas que ya demostramos. Tomemos como ejemplo el teorema de absorción.

TEOREMA 6a): $x + xy = x$.

$x + xy = x \cdot 1 + xy$	por el postulado:	2b)
$= x(1 + y)$		4a)
$= x(y + 1)$		3a)
$= x \cdot 1$		2a)
$= x$		2b)

TEOREMA 6b): $x(x + y) = x$ por dualidad.

Es posible demostrar los teoremas del álgebra booleana utilizando tablas de verdad. En esas tablas, se verifica que ambos miembros de la relación den resultados idénticos para todas las posibles combinaciones de las variables que intervienen. La siguiente tabla de verdad verifica el primer teorema de absorción.

x	y	xy	$x + xy$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Las demostraciones algebraicas de la ley asociativa y del teorema de DeMorgan son largas y no se presentarán aquí. Sin embargo, es fácil mostrar su validez con tablas de verdad. Por ejemplo, he aquí la tabla de verdad del primer teorema de DeMorgan $(x + y)' = x'y'$.

x	y	$x + y$	$(x + y)'$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

x'	y'	$x'y'$
1	1	1
1	0	0
0	1	0
0	0	0

Precedencia de operadores

La precedencia de operadores para evaluar expresiones booleanas es 1) paréntesis, 2) NOT, 3) AND y 4) OR. Dicho de otro modo, la expresión encerrada en paréntesis se debe evaluar antes que todas las demás operaciones. La siguiente operación que tiene precedencia es el complemento, seguida del AND y por último el OR. Por ejemplo, consideremos la tabla de verdad para el teorema de DeMorgan. El miembro izquierdo de la expresión es $(x + y)'$. Por tanto, la expresión dentro de los paréntesis se evalúa primero y luego se complementa el resultado. El miembro derecho de la expresión es $x'y'$. Por tanto, se evalúan primero el complemento de x y el complemento de y, y luego se obtiene el AND del resultado. Cabe señalar que rige la misma precedencia en la aritmética ordinaria (excepto el complemento) si sustituimos la multiplicación y la suma por el AND y el OR, respectivamente.

2-4 FUNCIONES BOOLEANAS

El álgebra booleana es un álgebra que se ocupa de variables binarias y operaciones lógicas. Una función booleana descrita por una expresión algebraica consta de variables binarias, las constantes 0 y 1, y los símbolos lógicos de operación. Para un valor dado de las variables binarias, la función puede ser igual a 1 o bien a 0. Considere por ejemplo esta función booleana:

$$F_1 = x + y'z$$

La función F_1 es igual a 1 si x es igual a 1 o si tanto y' como z son iguales a 1. F_1 es igual a 0 en todos los demás casos. La operación de complemento hace que si $y' = 1$, entonces $y = 0$. Por tanto, podemos decir que $F_1 = 1$ si $x = 1$ o si $y = 0$ y $z = 1$. Una función booleana expresa la relación lógica entre variables binarias. Se evalúa determinando el valor binario de la expresión para todos los posibles valores de las variables.

Podemos representar una función booleana en una tabla de verdad. Una tabla de verdad es una lista de combinaciones de unos y ceros asignados a las variables binarias y una columna que muestra el valor de la función para cada combinación binaria. El número de filas de la tabla es 2^n, donde n es el número de variables de la función. Las combinaciones binarias para la tabla de verdad se obtienen de los números binarios, contando de 0 hasta $2^n - 1$. La tabla 2-2 muestra la tabla de verdad para la función F_1. Hay ocho posibles combinaciones binarias para asignar bits a las tres variables x, y y z. La columna rotulada F_1 contiene 0 o 1 para cada una

Tabla 2-2
Tablas de verdad para F_1 y F_2

x	y	z	F_1	F_2
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	0
1	1	1	1	0

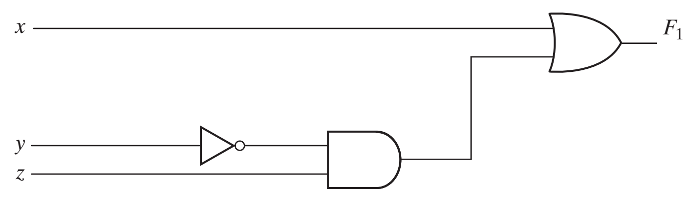


FIGURA 2-1
Implementación de $F_1 = x + y'z$ con compuertas

de esas combinaciones. La tabla indica que la función es igual a 1 cuando $x = 1$ o cuando $yz = 01$. En los demás casos, es igual a 0.

Una función booleana se puede transformar de una expresión algebraica a un diagrama de circuitos hecho con compuertas lógicas. En la figura 2-1 se presenta el diagrama de circuito lógico para F_1. Hay un inversor para generar el complemento de la entrada y, una compuerta AND para el término $y'z$ y una compuerta OR que combina los dos términos. En los diagramas de circuitos lógicos, las variables de la función son las entradas del circuito, y la variable binaria F_1 es la salida del circuito.

Sólo hay una forma de representar una función booleana en una tabla de verdad. En cambio, cuando la función está en forma algebraica, puede expresarse de varias maneras. La expresión específica empleada para designar la función también determinará la interconexión de compuertas en el diagrama de circuito lógico. Manipulando una expresión booleana según las reglas del álgebra booleana, a veces es posible obtener una expresión más simple para la misma función y así reducir el número de compuertas del circuito y el número de entradas de las compuertas. Consideremos, por ejemplo, esta función booleana:

$$F_2 = x'y'z + x'yz + xy'$$

La implementación de esta función con compuertas lógicas se muestra en la figura 2-2a). Las variables de entrada x y y se complementan con inversores para obtener x' y y'. Los tres términos de la expresión se implementan con tres compuertas AND. La compuerta OR forma el OR lógico de los tres términos. La tabla de verdad para F_2 se presenta en la tabla 2-2. La fun-

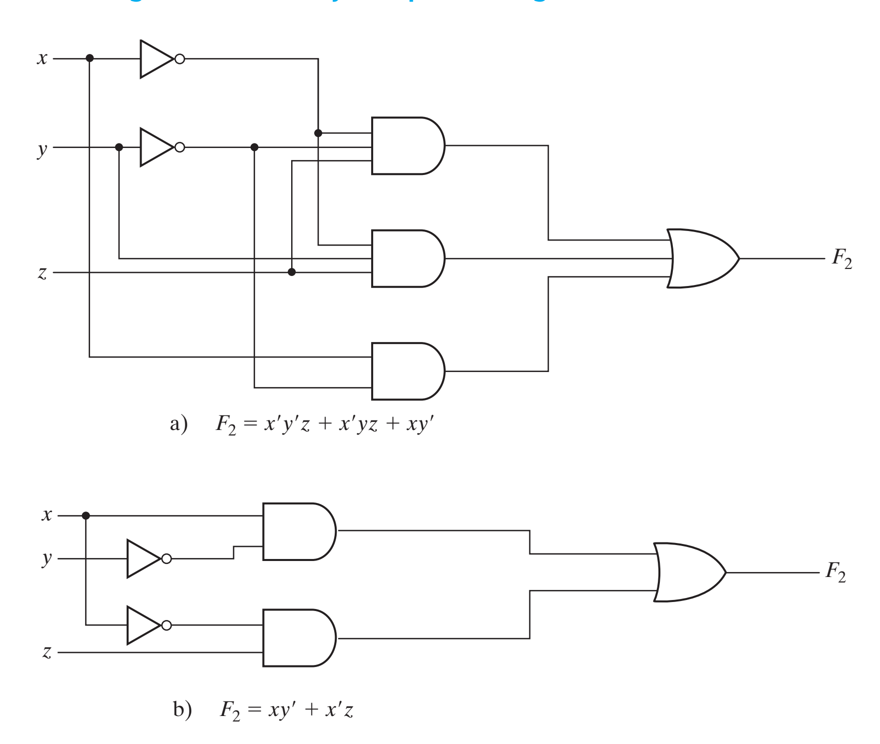


FIGURA 2-2
Implementación de la función booleana F_2 con compuertas

ción es igual a 1 cuando $xyz = 001$ o 011, o cuando $xy = 10$ (sin importar el valor de z); en los demás casos es igual a 0. Esto produce cuatro unos y cuatro ceros para F_2.

Consideremos ahora la posible simplificación de la función aplicando algunas de las identidades del álgebra booleana:

$$F_2 = x'y'z + x'yz + xy' = x'z(y' + y) + xy' = x'z + xy'$$

La función se reduce a únicamente dos términos y se puede implementar con compuertas como se indica en la figura 2-2b). Es evidente que el circuito de b) es más sencillo que el de a), aunque ambos realizan la misma función. Podemos usar una tabla de verdad para comprobar que las dos expresiones son equivalentes. La expresión simplificada es igual a 1 cuando $xz = 01$ o cuando $xy = 10$. Esto produce los mismos cuatro unos en la tabla de verdad. Puesto que ambas expresiones producen la misma tabla de verdad, decimos que son equivalentes. Por tanto, los dos circuitos tienen las mismas salidas para todas las posibles combinaciones binarias de las tres variables de entrada. Ambas realizan la misma función, pero la que tiene menos compuertas y menos entradas a las compuertas es preferible porque requiere menos alambres y componentes.

Manipulación algebraica

Cuando se implementa una expresión booleana con compuertas lógicas, cada término requiere una compuerta y cada variable dentro del término implica una entrada a la compuerta. Definimos una *literal* como una sola variable dentro de un término, que podría estar complementa-

da o no. La función de la figura 2-2a) tiene tres términos y ocho literales; la de la figura 2-2b) tiene dos términos y cuatro literales. Si reducimos el número de términos, el número de literales, o ambas cosas, en una expresión booleana, podría obtenerse un circuito más sencillo. La manipulación del álgebra booleana consiste en su mayor parte en reducir una expresión con objeto de obtener un circuito más simple. Las funciones de hasta cinco variables se pueden simplificar con el método de mapa que se describirá en el capítulo siguiente. En el caso de funciones booleanas complejas, los diseñadores digitales utilizan programas de minimización computarizados. El único método manual con que se cuenta es un procedimiento de recortes y ensayos que utiliza las relaciones básicas y otras técnicas de manipulación que con el uso se vuelven familiares. Los ejemplos que siguen ilustran la manipulación algebraica del álgebra booleana.

EJEMPLO 2-1

Simplifique las funciones booleanas siguientes al número mínimo de literales.

1. $x(x' + y) = xx' + xy = 0 + xy = xy.$
2. $x + x'y = (x + x')(x + y) = 1(x + y) = x + y.$
3. $(x + y)(x + y') = x + xy + xy' + yy' = x(1 + y + y') = x.$
4. $xy + x'z + yz = xy + x'z + yz(x + x')$
   $= xy + x'z + xyz + x'yz$
   $= xy(1 + z) + x'z(1 + y)$
   $= xy + x'z.$
5. $(x + y)(x' + z)(y + z) = (x + y)(x' + z)$ por dualidad de la función 4.

■

Las funciones 1 y 2 son una la dual de la otra y utilizan expresiones duales en pasos correspondientes. Una forma más sencilla de simplificar la función 3 es con el postulado 4b) de la tabla 2-1: $(x + y)(x + y') = x + yy' = x$. La cuarta función ilustra el hecho de que un aumento en el número de literales a veces da pie a una expresión final más simple. La función 5 no se minimiza directamente, pero puede deducirse del dual de los pasos empleados para deducir la función 4. Las funciones 4 y 5 llevan el nombre de *teorema de consenso*.

Complemento de una función

El complemento de una función F es F' y se obtiene intercambiando ceros por unos y unos por ceros en el valor de F. El complemento de una función podría deducirse algebraicamente empleando el teorema de DeMorgan. Este par de teoremas se presenta en la tabla 2-1 para dos variables. Los teoremas de DeMorgan se pueden extender a tres o más variables. La forma de tres variables del primer teorema de DeMorgan se deduce como sigue, utilizando postulados y teoremas de la tabla 2-1:

$(A + B + C)'$	$= (A + x)'$	sea $B + C = x$
	$= A'x'$	por el teorema 5a) (DeMorgan)
	$= A'(B + C)'$	sustituimos $B + C = x$
	$= A'(B'C')$	por el teorema 5a) (DeMorgan)
	$= A'B'C'$	por el teorema 4b) (asociatividad)

Los teoremas de DeMorgan para cualquier número de variables tienen una forma similar al caso de dos variables y se deducen por sustituciones sucesivas como se hizo en la deducción anterior. Estos teoremas se pueden generalizar así:

$$(A + B + C + D + \ldots + F)' = A'B'C'D' \ldots F'$$
$$(ABCD \ldots F)' = A' + B' + C' + D' + \ldots + F'$$

La forma generalizada del teorema de DeMorgan dice que el complemento de una función se obtiene intercambiando operadores AND y OR y complementando cada literal.

EJEMPLO 2-2

Halle el complemento de las funciones $F_1 = x'yz' + x'y'z$ y $F_2 = x(y'z' + yz)$. Aplicando el teorema de DeMorgan tantas veces como sea necesario, se obtienen los complementos como sigue:

$$F_1' = (x'yz' + x'y'z)' = (x'yz')'(x'y'z)' = (x + y' + z)(x + y + z')$$
$$F_2' = [x(y'z' + yz)]' = x' + (y'z' + yz)' = x' + (y'z')'(yz)'$$
$$= x' + (y + z)(y' + z')$$

■

Un procedimiento más sencillo para deducir el complemento de una función consiste en obtener el dual de la función y complementar cada literal. Este método es consecuencia del teorema generalizado de DeMorgan. Recuerde que el dual de una función se obtiene intercambiando operadores AND y OR, y unos y ceros.

EJEMPLO 2-3

Determine el complemento de las funciones F_1 y F_2 del ejemplo 2-2 obteniendo sus duales y complementando cada literal.

1. $F_1 = x'yz' + x'y'z$.
El dual de F_1 es $(x' + y + z')(x' + y' + z)$.
Complementamos cada literal: $(x + y' + z)(x + y + z') = F_1'$.

2. $F_2 = x(y'z' + yz)$.
El dual de F_2 es $x + (y' + z')(y + z)$.
Complementamos cada literal: $x' + (y + z)(y' + z') = F_2'$.

■

2-5 FORMAS CANÓNICAS Y ESTÁNDAR

Minitérminos y maxitérminos

Una variable binaria podría aparecer en su forma normal (x) o en su forma complementada (x'). Considere ahora dos variables binarias x y y que se combinan con una operación AND. Puesto que cada variable podría aparecer en cualquiera de sus dos formas, hay cuatro combinaciones posibles: $x'y'$, $x'y$, xy' y xy. Cada uno de estos cuatro términos AND es un *minitérmi-*

Tabla 2-3
Minitérminos y maxitérminos para tres variables binarias

			Minitérminos		Maxitérminos	
x	y	z	**Términos**	**Designación**	**Términos**	**Designación**
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

no, o *producto estándar*. De manera similar, podemos combinar n variables para formar 2^n minitérminos. Éstos podrían obtenerse con un método similar al que se muestra en la tabla 2-3 para tres variables. Se enumeran los números binarios de 0 a $2^n - 1$ bajo las n variables. Cada minitérmino se obtiene de un término AND de las n variables, poniendo un apóstrofo a cada variable si el bit correspondiente del número binario es un 0 y sin apóstrofo si es un 1. En la tabla también se muestra un símbolo para cada minitérmino, el cual tiene la forma m_j, donde j denota el equivalente decimal del número binario del minitérmino designado.

Asimismo, n variables que forman un término OR, donde cada variable puede tener apóstrofo o no, dan pie a 2^n posibles combinaciones, llamadas *maxitérminos* o *sumas estándar*. En la tabla 2-3 se presentan los ocho maxitérminos de tres variables, junto con su designación simbólica. Podemos obtener de manera similar cualesquier 2^n maxitérminos para n variables. Cada maxitérmino se obtiene de un término OR de las n variables, donde cada variable lleva un apóstrofo si el bit correspondiente del número binario es 1. Cabe señalar que cada maxitérmino es el complemento de su minitérmino correspondiente, y viceversa.

Se puede expresar algebraicamente una función booleana a partir de una tabla de verdad dada formando un minitérmino para cada combinación de las variables que produce un 1 en la función, y formando después el OR de todos esos términos. Por ejemplo, la función f_1 de la

Tabla 2-4
Funciones de tres variables

x	y	z	Función f_1	Función f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

tabla 2-4 se obtiene expresando las combinaciones 001, 100 y 111 como $x'y'z$, $xy'z'$ y xyz, respectivamente. Puesto que cada uno de estos minitérminos hace que $f_1 = 1$, tenemos

$$f_1 = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7$$

De forma similar, es posible verificar fácilmente que

$$f_2 = x'yz + xy'z + xyz' + xyz = m_3 + m_5 + m_6 + m_7$$

Estos ejemplos ilustran una propiedad importante del álgebra booleana: cualquier función booleana se puede expresar como una suma de minitérminos (donde "suma" se refiere al OR de los términos).

Considere ahora el complemento de una función booleana. Podría leerse de la tabla de verdad formando un minitérmino para cada combinación que produce un 0 en la función, y haciendo después el OR de esos términos. El complemento de f_1 se lee así:

$$f'_1 = x'y'z' + x'yz' + x'yz + xy'z + xyz'$$

Si obtenemos el complemento de f'_1, obtendremos la función f_1:

$$\begin{aligned} f_1 &= (x + y + z)(x + y' + z)(x' + y + z')(x' + y' + z) \\ &= M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6 \end{aligned}$$

Asimismo, podemos leer la expresión para f_2 de la tabla:

$$\begin{aligned} f_2 &= (x + y + z)(x + y + z')(x + y' + z)(x' + y + z) \\ &= M_0 M_1 M_2 M_4 \end{aligned}$$

Estos ejemplos ilustran una segunda propiedad del álgebra booleana: cualquier función booleana se puede expresar como un producto de maxitérminos (donde "producto" se refiere a hacer el AND de los términos). El procedimiento para obtener el producto de maxitérminos directamente de la tabla de verdad es el siguiente. Se forma un maxitérmino para cada combinación de las variables que produce un 0 en la función y luego se hace el AND de todos esos maxitérminos. Se dice que las funciones booleanas expresadas como suma de minitérminos o producto de maxitérminos están en *forma canónica*.

Suma de minitérminos

Dijimos antes que, para n variables binarias, podemos obtener 2^n minitérminos distintos, y que es posible expresar cualquier función booleana como una suma de minitérminos. Los minitérminos cuya suma define a la función booleana son los que producen los unos de la función en una tabla de verdad. Puesto que la función puede dar 1 o 0 con cada minitérmino, y dado que hay 2^n minitérminos, podemos calcular que el número de funciones que es posible formar con n variables es 2^{2^n}. A veces es útil expresar la función booleana en su forma de suma de minitérminos. Si no está ya en esa forma, esto se logra expandiendo primero la expresión a una suma de términos AND. Luego se examina cada término para ver si contiene todas las variables. Si falta una o más variables, se le hace AND con una expresión como $x + x'$, donde x es una de las variables faltantes. El ejemplo que sigue aclarará el procedimiento.

EJEMPLO 2-4

Exprese la función booleana $F = A + B'C$ como suma de minitérminos. La función tiene tres variables, A, B y C. En el primer término, A, faltan dos variables; por tanto:

$$A = A(B + B') = AB + AB'$$

A esta función todavía le falta una variable:

$$\begin{aligned} A &= AB(C + C') + AB'(C + C') \\ &= ABC + ABC' + AB'C + AB'C' \end{aligned}$$

Al segundo término, $B'C$, le falta una variable:

$$B'C = B'C(A + A') = AB'C + A'B'C$$

Al combinar todos los términos, tenemos

$$\begin{aligned} F &= A + B'C \\ &= ABC + ABC' + AB'C + AB'C' + A'B'C \end{aligned}$$

Sin embargo, $AB'C$ aparece dos veces y, según el teorema 1 ($x + x = x$), podemos eliminar uno de ellos. Después de reacomodar los minitérminos en orden ascendente, obtenemos por fin

$$\begin{aligned} F &= A'B'C + AB'C' + AB'C + ABC' + ABC \\ &= m_1 + m_4 + m_5 + m_6 + m_7 \end{aligned}$$

■

Hay ocasiones en que conviene expresar la función booleana, en su forma de suma de minitérminos, con la siguiente notación abreviada:

$$F(A, B, C) = \sum(1, 4, 5, 6, 7)$$

El símbolo de sumatoria, $\sum$, representa el OR de los términos; los números que le siguen son los minitérminos de la función. Las letras entre paréntesis después de F son una lista de las variables en el orden que se usará al convertir un minitérmino en un término AND.

Otro procedimiento para obtener los minitérminos de una función booleana consiste en deducir la tabla de verdad directamente de la expresión algebraica y luego leer los minitérminos de esa tabla. Considere la función booleana del ejemplo 2-4:

$$F = A + B'C$$

La tabla de verdad que se muestra en la tabla 2-5 se deduce directamente de la expresión algebraica enumerando las ocho combinaciones binarias bajo las variables A, B y C e insertan-

Tabla 2-5
Tabla de verdad para $F = A + B'C$

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

do unos bajo F para las combinaciones en las que $A = 1$ y $BC = 01$. Luego, se lee en la tabla de verdad que los cinco minitérminos de la función son 1, 4, 5, 6 y 7.

Producto de maxitérminos

Cada una de las 2^{2n} funciones de n variables binarias se puede expresar también como un producto de maxitérminos. Para expresar la función booleana como producto de maxitérminos, primero debe ponerse en formato de términos OR. Esto podría hacerse con la ayuda de la ley distributiva, $x + yz = (x + y)(x + z)$. Luego, se hace el OR de cualquier variable faltante x en cada término OR con xx'. El ejemplo que sigue aclarará el procedimiento.

EJEMPLO 2-5

Exprese la función booleana $F = xy + x'z$ en forma de producto de maxitérminos. Primero, se convierte la función en términos OR empleando la ley distributiva:

$$\begin{aligned} F &= xy + x'z = (xy + x')(xy + z) \\ &= (x + x')(y + x')(x + z)(y + z) \\ &= (x' + y)(x + z)(y + z) \end{aligned}$$

La función tiene tres variables, x, y y z. A cada término OR le falta una variable; por tanto:

$$\begin{aligned} x' + y &= x' + y + zz' = (x' + y + z)(x' + y + z') \\ x + z &= x + z + yy' = (x + y + z)(x + y' + z) \\ y + z &= y + z + xx' = (x + y + z)(x' + y + z) \end{aligned}$$

Después de combinar todos los términos y eliminar los que aparecen más de una vez, se obtiene:

$$\begin{aligned} F &= (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z') \\ &= M_0 M_2 M_4 M_5 \end{aligned}$$

Una forma cómoda de expresar esta función es:

$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

El símbolo de producto, Π, denota el AND de maxitérminos; los números son los maxitérminos de la función.

■

Conversión entre formas canónicas

El complemento de una función expresado como la suma de minitérminos es igual a la suma de los minitérminos que faltan en la función original. Ello se debe a que la función original se expresa con los minitérminos que hacen que la función sea igual a 1, mientras que su complemento es 1 para aquellos minitérminos que hacen que la función original sea 0. Por ejemplo, considere la función

$$F(A, B, C) = \sum(1, 4, 5, 6, 7)$$

Su complemento se expresa como

$$F'(A, B, C) = \sum(0, 2, 3) = m_0 + m_2 + m_3$$

Ahora bien, si se determina el complemento de F' por el teorema de DeMorgan, se obtiene F en una forma distinta:

$$F = (m_0 + m_2 + m_3)' = m_0' \cdot m_2' \cdot m_3' = M_0 M_2 M_3 = \Pi\,(0, 2, 3)$$

Esta última conversión es consecuencia de la definición de minitérminos y maxitérminos, como se muestra en la tabla 2-3. En esa tabla es evidente que se cumple la relación

$$m_j' = M_j$$

Es decir, el maxitérmino con subíndice j es el complemento del minitérmino que lleva ese subíndice, y viceversa.

El último ejemplo ilustra la conversión entre una función expresada como suma de minitérminos y su equivalente en producto de maxitérminos. Un argumento similar demuestra que la conversión entre el producto de maxitérminos y la suma de minitérminos es similar. Ahora plantearemos un procedimiento general de conversión. Para convertir de una forma canónica a otra, intercambiamos los símbolos $\sum$ y Π e incluimos en la lista sólo los números que faltaban en la forma original. Para hallar los términos faltantes, debemos recordar que el número total de minitérminos o maxitérminos es 2^n, donde n es el número de variables binarias en la función.

Es posible convertir una función booleana de una expresión algebraica a un producto de maxitérminos utilizando una tabla de verdad y el procedimiento de conversión canónica. Considere, por ejemplo, la expresión booleana

$$F = xy + x'z$$

Primero, obtenemos la tabla de verdad de la función, la cual se muestra en la tabla 2-6. Los unos bajo F en la tabla se determinan a partir de las combinaciones de las variables en las que $xy = 11$ o $xz = 01$. De la tabla de verdad deducimos que los minitérminos de la función son 1, 3, 6 y 7. La función expresada como suma de minitérminos es

$$F(x, y, z) = \sum(1, 3, 6, 7)$$

Puesto que en total hay ocho minitérminos o maxitérminos en una función de tres variables, deducimos que los términos faltantes son 0, 2, 4 y 5. La función expresada como producto de maxitérminos es

$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

Ésta es la misma respuesta que obtuvimos en el ejemplo 2-5.

Tabla 2-6
Tabla de verdad para $F = xy + x'z$

x	*y*	*z*	*F*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Formas estándar

Las dos formas canónicas del álgebra booleana son formas básicas que se obtienen al leer una función de su tabla de verdad, pero casi nunca son las que tienen el número mínimo de literales, porque cada minitérmino o maxitérmino debe contener, por definición, *todas* las variables, sea complementadas o sin complementar.

Otra forma de expresar funciones booleanas es en forma *estándar*. En esta configuración, los términos que forman la función podrían contener una, dos o cualquier número de literales. Hay dos tipos de formas estándar: la suma de productos y el producto de sumas.

La *suma de productos* es una expresión booleana que contiene términos AND, llamados *términos de producto*, con una o más literales cada uno. La *suma* denota el OR de esos términos. Un ejemplo de función expresada como suma de productos es

$$F_1 = y' + xy + x'yz'$$

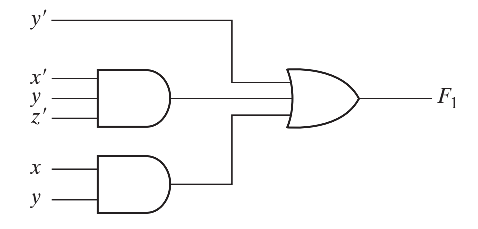
La expresión tiene tres términos de producto con una, dos y tres literales. Su suma es realmente una operación OR.

El diagrama de lógica de una expresión de suma de productos consiste en un grupo de compuertas AND seguidas de una sola compuerta OR. Este patrón de configuración se muestra en la figura 2-3a). Cada término de producto requiere una compuerta AND, salvo los términos que sólo tienen una literal. La suma lógica se forma con una compuerta OR cuyas entradas son las salidas de las compuertas AND y las literales solas. Suponemos que contamos directamente con las variables de entrada en forma de complemento, así que no se han incluido inversores en el diagrama. Esta configuración de circuito se denomina implementación de dos niveles.

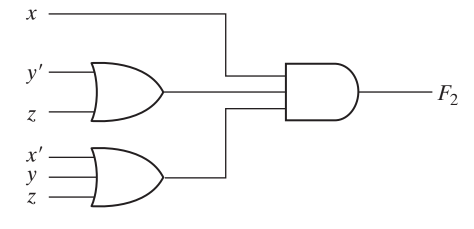
Un *producto de sumas* es una expresión booleana que contiene términos OR, llamados *términos de suma*. Cada término puede tener cualquier cantidad de literales. El *producto* denota el AND de esos términos. Un ejemplo de función expresada como producto de sumas es

$$F_2 = x(y' + z)(x' + y + z)$$

Esta expresión tiene tres términos de suma con una, dos y tres literales. El producto es una operación AND. El uso de las palabras *producto* y *suma* proviene de la similitud entre la operación AND y el producto aritmético (multiplicación), y de la similitud entre la operación OR y la suma aritmética (adición). La estructura de compuertas de la expresión de producto de sumas consiste en un grupo de compuertas OR para los términos de suma (excepto la literal



a) Suma de productos



b) Producto de sumas

FIGURA 2-3
Implementación de dos niveles

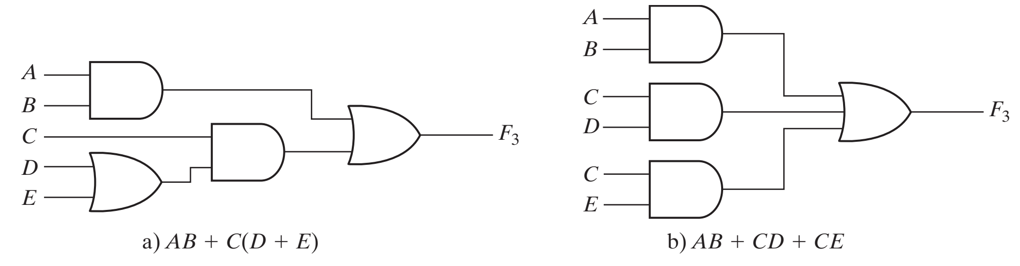


FIGURA 2-4
Implementación de tres y dos niveles

sola) seguidas de una compuerta AND. Esto se observa en la figura 2-3b). Este tipo estándar de expresión produce una estructura de compuertas de dos niveles.

Las funciones booleanas se pueden expresar en forma no estándar. Por ejemplo, la función

$$F_3 = AB + C(D + E)$$

no es una suma de productos ni un producto de sumas. Su implementación se indica en la figura 2-4a), y requiere dos compuertas AND y dos compuertas OR. Este circuito tiene tres niveles de compuertas, y puede transformarse a una forma estándar utilizando la ley distributiva para eliminar los paréntesis:

$$F_3 = AB + C(D + E) = AB + CD + CE$$

La expresión de suma de productos se implementa en la figura 2-4b). En general, es preferible una implementación de dos niveles porque produce el mínimo de retardo en compuertas cuando la señal se propaga de las entradas a la salida.

2-6 OTRAS OPERACIONES LÓGICAS

Cuando colocamos los operadores binarios AND y OR entre dos variables, x y y, forman dos funciones booleanas, $x \cdot y$ y $x + y$, respectivamente. Se ha señalado ya que hay 2^{2n} funciones para n variables binarias. En el caso de dos variables, $n = 2$, y el número de posibles funciones booleanas es 16. Por tanto, las funciones AND y OR son sólo dos de un total de 16 posibles funciones que se forman con dos variables binarias. Sería interesante encontrar las otras 14 funciones e investigar sus propiedades.

En la tabla 2-7 se presentan las tablas de verdad para las 16 funciones que se forman con dos variables binarias, x y y. Cada una de las 16 columnas, F_0 a F_{15}, representa una tabla de verdad de una posible función de las dos variables x y y. Observe que las funciones se determinan a partir de las 16 combinaciones binarias que se pueden asignar a F. Las 16 funciones se expresan algebraicamente con funciones booleanas, como se indica en la primera columna de la tabla 2-8. Las expresiones booleanas que se incluyen se han simplificado al número mínimo de literales.

Aunque cada función se puede expresar en términos de los operadores booleanos AND, OR y NOT, no hay motivo para no asignar símbolos de operador especiales que expresen las otras funciones. Dichos símbolos aparecen en la segunda columna de la tabla 2-8. Sin embargo, los

Tabla 2-7
Tablas de verdad para las 16 funciones de dos variables binarias

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

diseñadores digitales no usan comúnmente ninguno de los símbolos que se muestran, con excepción del símbolo de OR exclusivo, ⊕.

Cada una de las funciones de la tabla 2-8 se presenta acompañada de un nombre y un comentario que explica algo de la función. Las 16 funciones de la lista se subdividen en tres categorías:

1. Dos funciones que producen una constante, 0 o 1.
2. Cuatro funciones con operaciones unarias: complemento y transferencia.
3. Diez funciones con operadores binarios que definen ocho operaciones distintas: AND, OR, NAND, NOR, OR exclusivo, equivalencia, inhibición e implicación.

Tabla 2-8
Expresiones booleanas para las 16 funciones de dos variables

Funciones booleanas	Símbolo de operador	Nombre	Comentarios
$F_0 = 0$		Nula	Constante binaria 0
$F_1 = xy$	$x \cdot y$	AND	x y y
$F_2 = xy'$	x/y	Inhibición	x, pero no y
$F_3 = x$		Transferencia	x'
$F_4 = x'y$	y/x	Inhibición	y, pero no x
$F_5 = y$		Transferencia	y
$F_6 = xy' + x'y$	$x \oplus y$	OR exclusivo	x o y, pero no ambas
$F_7 = x + y$	$x + y$	OR	x o y
$F_8 = (x + y)'$	$x \downarrow y$	NOR	No OR
$F_9 = xy + x'y'$	$(x \oplus y)'$	Equivalencia	x es igual a y
$F_{10} = y'$	y'	Complemento	No y
$F_{11} = x + y'$	$x \subset y$	Implicación	Si y, entonces x
$F_{12} = x'$	x'	Complemento	No x
$F_{13} = x' + y$	$x \supset y$	Implicación	Si x, entonces y
$F_{14} = (xy)'$	$x \uparrow y$	NAND	No AND
$F_{15} = 1$		Identidad	Constante binaria 1

Las constantes de funciones booleanas sólo pueden ser iguales a 1 o a 0. La función de complemento produce el complemento de cada una de las variables binarias. Se llama *transferencia* a toda función que es igual a una variable de entrada, porque la variable, x o y, se transfiere a través de la compuerta que forma la función sin alterar su valor. De los ocho operadores binarios, dos (inhibición e implicación) se usan en lógica, pero casi nunca en lógica de computadoras. Ya mencionamos los operadores AND y OR en relación con el álgebra booleana. Las otras cuatro funciones se usan extensamente en el diseño de sistemas digitales.

La función NOR es el complemento de la función OR y su nombre es la abreviatura de *no* OR. Asimismo, NAND es el complemento de AND y es la abreviatura de *no* AND. El OR exclusivo, que se abrevia XOR, es similar al OR, pero excluye la combinación en que *tanto x como y* son 1. La equivalencia es una función que es 1 cuando las dos variables binarias son iguales, es decir, cuando ambas son 0 o cuando ambas son 1. Las funciones de OR exclusivo y equivalencia son una el complemento de la otra. Esto se comprueba fácilmente inspeccionando la tabla 2-7. La tabla de verdad para el OR exclusivo es F_6, y para la equivalencia, F_9, y estas dos funciones son una el complemento de la otra. Por ello, llamamos NOR exclusivo a la función de equivalencia, y la abreviamos XNOR.

El álgebra booleana, tal como se definió en la sección 2-2, tiene dos operadores binarios, a los que hemos llamado AND y OR, y un operador unario, NOT (complemento). De las definiciones, hemos deducido varias propiedades de esos operadores y ahora hemos definido otros operadores binarios en términos de los primeros. Nada tiene de singular tal procedimiento. Bien podríamos haber partido del operador NOR ($\downarrow$), por ejemplo, y posteriormente haber definido AND, OR y NOT en términos de él. No obstante, hay motivos de peso para introducir el álgebra booleana de la manera en que lo hicimos. Los conceptos de "y", "o" y "no" son conocidos y la gente los usa para expresar ideas lógicas cotidianas. Además, los postulados de Huntington reflejan la naturaleza dual del álgebra y hacen hincapié en la simetría mutua de $+$ y $\cdot$.

2-7 COMPUERTAS LÓGICAS DIGITALES

Puesto que las funciones booleanas se expresan en términos de operaciones AND, OR y NOT, es más fácil implementar una función booleana con estos tipos de compuertas. La posibilidad de construir compuertas para las otras operaciones lógicas tiene interés práctico. Los factores a considerar al investigar la construcción de otros tipos de compuertas lógicas son: 1) la factibilidad y economía de producir la compuerta con componentes físicos, 2) la posibilidad de extender la compuerta a más de dos entradas, 3) las propiedades básicas del operador binario, como conmutatividad y asociatividad, y 4) la capacidad de la compuerta para implementar funciones booleanas solas o junto con otras compuertas.

De las 16 funciones definidas en la tabla 2-8, dos son iguales a una constante y cuatro se repiten dos veces. Sólo quedan diez funciones que considerar como candidatas para compuertas lógicas. Dos —inhibición e implicación— no son conmutativas ni asociativas, por lo que no resulta práctico su uso como compuertas lógicas estándar. Las otras ocho: complemento, transferencia, AND, OR, NAND, NOR, OR exclusivo y equivalencia se emplean como compuertas estándar en diseño digital.

Los símbolos gráficos y tablas de verdad de las ocho compuertas aparecen en la figura 2-5. Cada compuerta tiene una o dos variables binarias de entrada designadas con x y y, y una variable binaria de salida designada con F. Ya definimos los circuitos de AND, OR y el inversor

Nombre	Símbolo gráfico	Función algebraica	Tabla de verdad		
			x	y	F
AND	x, y → F	$F = xy$	0	0	0
			0	1	0
			1	0	0
			1	1	1
			x	y	F
OR	x, y → F	$F = x + y$	0	0	0
			0	1	1
			1	0	1
			1	1	1
			x		F
Inversor	x → F	$F = x'$	0		1
			1		0
			x		F
Búfer	x → F	$F = x$	0		0
			1		1
			x	y	F
NAND	x, y → F	$F = (xy)'$	0	0	1
			0	1	1
			1	0	1
			1	1	0
			x	y	F
NOR	x, y → F	$F = (x + y)'$	0	0	1
			0	1	0
			1	0	0
			1	1	0
			x	y	F
OR exclusivo (XOR)	x, y → F	$F = xy' + x'y = x \oplus y$	0	0	0
			0	1	1
			1	0	1
			1	1	0
			x	y	F
NOR exclusivo o equivalencia	x, y → F	$F = xy + x'y' = (x \oplus y)'$	0	0	1
			0	1	0
			1	0	0
			1	1	1

FIGURA 2-5
Compuertas lógicas digitales

en la figura 1-6. El circuito inversor invierte el sentido lógico de una variable binaria: produce la función NOT, o complemento. El pequeño círculo en la salida del símbolo gráfico de un inversor (llamado *burbuja*) indica el complemento lógico. El símbolo de triángulo, por sí solo, denota un circuito búfer. Un búfer produce la función de *transferencia*, pero no una operación lógica, ya que el valor binario de la salida es igual al valor binario de la entrada. Este circuito sirve para amplificar la potencia de la señal y equivale a dos inversores conectados en cascada.

La función NAND es el complemento de la función AND, como lo indica el símbolo gráfico que consiste en un símbolo gráfico AND seguido de una burbuja. La función NOR es el complemento de la función OR y su símbolo gráfico es el de OR seguido de una burbuja. Las compuertas NAND y NOR se usan mucho como compuertas lógicas estándar y, de hecho, son mucho más populares que las compuertas AND y OR. Ello se debe a que es fácil construir compuertas NAND y NOR con circuitos de transistores, y a que es fácil implementar con ellas circuitos digitales.

La compuerta de OR exclusivo tiene un símbolo gráfico parecido al de la compuerta OR, sólo que lleva una línea curva adicional del lado de la entrada. La compuerta de equivalencia, o NOR exclusivo, es el complemento del OR exclusivo, como indica la burbuja en el lado de salida del símbolo gráfico.

Extensión a múltiples entradas

Las compuertas que se muestran en la figura 2-5 —con excepción del inversor y el búfer— se pueden extender de modo que tengan más de dos entradas. Es posible extender una compuerta a múltiples entradas si la operación binaria que representa es conmutativa y asociativa. Las operaciones AND y OR, definidas en el álgebra booleana, poseen esas dos propiedades. Para la función OR, tenemos

$$x + y = y + x \qquad \text{(conmutatividad)}$$

y

$$(x + y) + z = x + (y + z) = x + y + z \qquad \text{(asociatividad),}$$

lo que nos dice que las entradas de la compuerta son intercambiables y que la función OR se puede extender a tres o más variables.

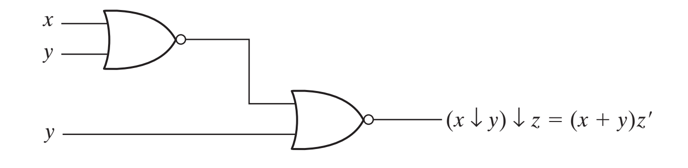
Las funciones NAND y NOR son conmutativas, y sus compuertas se extienden a más de dos entradas, si se modifica ligeramente la definición de la operación. El problema radica en que los operadores NAND y NOR no son asociativos [es decir, $(x \downarrow y) \downarrow z \neq x \downarrow (y \downarrow z)$], como se indica en la figura 2-6 y en las ecuaciones siguientes:

$$(x \downarrow y) \downarrow z = [(x + y)' + z]' = (x + y)z' = xz' + yz'$$
$$x \downarrow (y \downarrow z) = [x + (y + z)']' = x'(y + z) = x'y + x'z$$

Para superar este problema, definimos la compuerta NOR (o NAND) múltiple como una compuerta OR (o AND) complementada. Así, por definición, tenemos

$$x \downarrow y \downarrow z = (x + y + z)'$$
$$x \uparrow y \uparrow z = (xyz)'$$

Los símbolos gráficos para las compuertas de tres entradas se incluyen en la figura 2-7. Al escribir operaciones NOR y NAND en cascada, hay que usar los paréntesis correctos para indi-



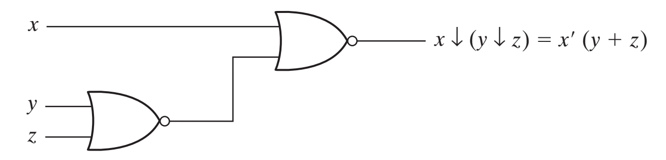
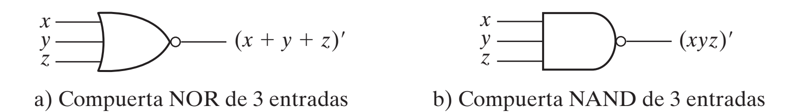
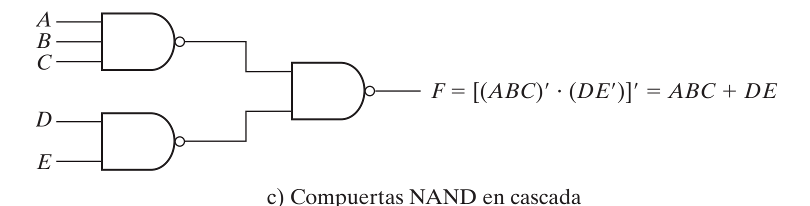


FIGURA 2-6
Demostración de la no asociatividad del operador NOR; $(x \downarrow y) \downarrow z \neq x \downarrow (y \downarrow z)$





c) Compuertas NAND en cascada

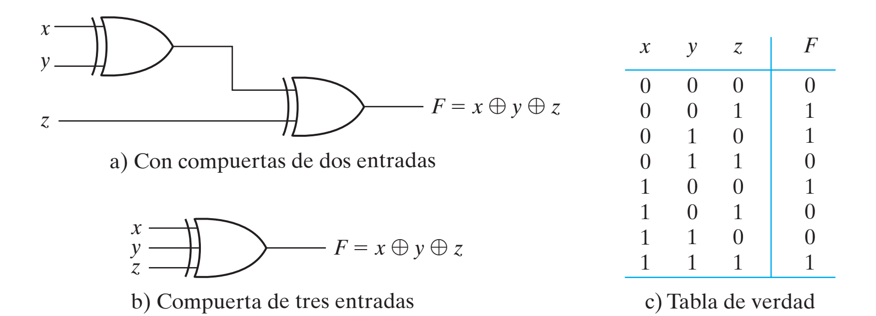
FIGURA 2-7
Compuertas NOR y NAND con múltiples entradas y en cascada

car el orden en que deben ir las compuertas. Para demostrar esto, consideremos el circuito de la figura 2-7c). La función booleana del circuito se escribe así:

$$F = \left[(ABC)'(DE)'\right]' = ABC + DE$$

La segunda expresión se obtiene del teorema de DeMorgan, y también demuestra que una expresión en forma de suma de productos se puede implementar con compuertas NAND. En la sección 3-6 trataremos más a fondo las compuertas NAND y NOR.

Las compuertas OR exclusivo y de equivalencia son tanto conmutativas como asociativas y se pueden extender a más de dos entradas. No obstante, las compuertas OR exclusivo de varias entradas son poco comunes en hardware. De hecho, incluso la función de dos entradas suele construirse con otros tipos de compuertas. Además, es preciso modificar la definición de la función al extenderla a más de dos variables. El OR exclusivo es una función *impar*, es decir, es igual a 1 si las variables de entrada tienen un número impar de unos. En la figura 2-8 se representa la construcción de una función OR exclusivo de tres entradas,



x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

c) Tabla de verdad

FIGURA 2-8
Compuerta OR exclusivo de tres entradas

aunque normalmente se la implementa conectando en cascada compuertas de dos entradas, como se observa en a). Gráficamente el OR exclusivo se representa con una sola compuerta de tres entradas, como en b). La tabla de verdad de c) indica claramente que la salida de F es igual a 1 si sólo una entrada es 1 o si las tres entradas son 1, es decir, si el número total de unos en las variables de entrada es *impar*. En la sección 3-8 se estudiará más a fondo el OR exclusivo.

Lógica positiva y negativa

La señal binaria en las entradas y salidas de cualquier compuerta tiene uno de dos valores, excepto durante una transición. Un valor de señal representa el 1 lógico, y el otro, el 0 lógico. Puesto que se asignan dos valores de señal a dos valores lógicos, puede haber dos asignaciones distintas de nivel de señal a valor lógico, como se indica en la figura 2-9. El nivel de señal más alto se designa con H, y el más bajo, con L. Si escogemos el nivel alto H para representar el 1 lógico, estaremos definiendo un sistema de lógica positiva. Si escogemos el nivel bajo L para representar el 1 lógico, definiremos un sistema de lógica negativa. Los términos positiva y negativa son un tanto engañosos porque ambas señales podrían ser positivas, o ambas negativas. No son los valores reales de la señal lo que determina el tipo de lógica, sino más bien la asignación de valores lógicos a las amplitudes relativas de los dos niveles de señal.

Las compuertas digitales en hardware se definen en términos de valores de señal como H y L. Corresponde al usuario decidir si la polaridad de la lógica va a ser positiva o negativa. Con-

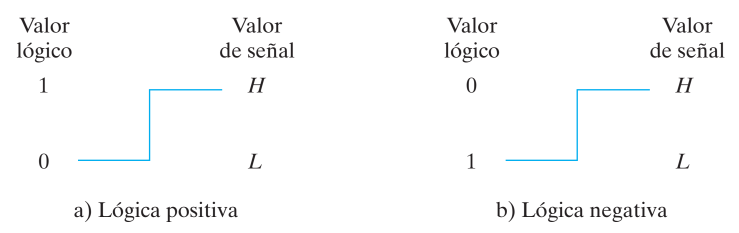
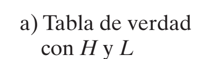
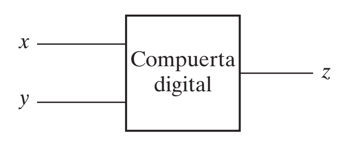


FIGURA 2-9
Asignación de señales y polaridad lógica

x	y	F
L	L	L
L	H	L
H	L	L
H	H	H



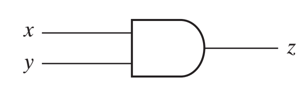
a) Tabla de verdad con H y L



b) Diagrama de bloque de compuerta

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

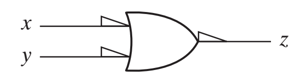
c) Tabla de verdad para lógica positiva



d) Compuerta AND de lógica positiva

x	y	z
1	1	1
1	0	1
0	1	1
0	0	0

e) Tabla de verdad para lógica negativa



f) Compuerta OR de lógica negativa

FIGURA 2-10
Demostración de lógica positiva y negativa

sideremos, por ejemplo, la compuerta electrónica que se muestra en la figura 2-10b). La tabla de verdad de esta compuerta se presenta en la figura 2-10a), y especifica el comportamiento físico de la compuerta cuando H es 3 volts y L es 0 volts. La tabla de verdad de la figura 2-10c) supone una asignación de lógica positiva, con $H = 1$ y $L = 0$. Esta tabla de verdad es igual a la de la operación AND. El símbolo gráfico para una compuerta AND con lógica positiva se muestra en la figura 2-10d).

Consideremos ahora la asignación de lógica negativa a la misma compuerta física, con $L = 1$ y $H = 0$. El resultado es la tabla de verdad de la figura 2-10e). Esta tabla representa la operación OR aunque las filas están invertidas. El símbolo gráfico para la compuerta OR de lógica negativa se aprecia en la figura 2-10f). Los pequeños triángulos en las entradas y la salida son *indicadores de polaridad*. La presencia de este indicador de polaridad en una terminal implica que se está suponiendo lógica negativa para la señal. Así, la misma compuerta física puede operar como compuerta AND de lógica positiva o como compuerta OR de lógica negativa.

La conversión de lógica positiva a lógica negativa, y viceversa, es básicamente una operación que cambia los unos a ceros y los ceros a unos tanto en las entradas como en la salida de la compuerta. Puesto que esta operación produce el dual de una función, el cambio de todas las terminales, de una polaridad a la otra, equivale a obtener el dual de la función. El resultado de esta conversión es que todas las operaciones AND se convierten en operaciones OR (o símbolos gráficos) y viceversa. Además, no debemos olvidarnos de incluir el triángulo indicador de polaridad en los símbolos gráficos cuando se supone lógica negativa. En este libro no usaremos compuertas de lógica negativa, y supondremos que todas las compuertas operan con una asignación de lógica positiva.

2-8 CIRCUITOS INTEGRADOS

Un circuito integrado (que se abrevia CI) es un cristal semiconductor de silicio, llamado *chip*, que contiene los componentes electrónicos para construir compuertas digitales. Las diversas compuertas se interconectan dentro del chip para formar el circuito requerido. El chip se monta en un recipiente de cerámica o plástico, y las conexiones se sueldan a terminales externas para formar el circuito integrado. El número de terminales podría variar desde 14 en un paquete de CI pequeño hasta varios miles en los paquetes más grandes. Cada CI tiene una designación numérica impresa en la superficie del paquete, para poder identificarlo. Los fabricantes proporcionan libros de datos, catálogos y sitios Web de Internet que contienen descripciones e información acerca de los CI que producen.

Niveles de integración

Los CI digitales suelen clasificarse según la complejidad de sus circuitos, la cual se mide por el número de compuertas lógicas incluidas en el paquete. La diferenciación entre los chips que tienen pocas compuertas internas y los que tienen cientos de miles de compuertas suele hacerse diciendo que el paquete es un dispositivo de integración a pequeña, mediana, gran o muy grande escala.

Los dispositivos de *integración a pequeña escala* (SSI, *small-scale integration*) contienen varias compuertas independientes en un solo paquete. Las entradas y salidas de las compuertas se conectan directamente a las terminales del paquete. El número de compuertas suele ser menor que 10 y está limitado por el número de terminales con que cuenta el CI.

Los dispositivos de *integración a mediana escala* (MSI, *medium-scale integration*) tienen una complejidad de entre 10 y 1000 compuertas en un solo paquete. Por lo regular, efectúan operaciones digitales elementales específicas. Presentaremos las funciones digitales de MSI en el capítulo 4 como decodificadores, sumadores y multiplexores, y en el capítulo 6, como registros y contadores.

Los dispositivos de *integración a gran escala* (LSI, *large-scale integration*) contienen miles de compuertas en un solo paquete. Incluyen sistemas digitales como procesadores, chips de memoria y dispositivos de lógica programable. Presentaremos algunos componentes LSI en el capítulo 8.

Los dispositivos de *integración a muy grande escala* (VLSI, *very large-scale integration*) contienen cientos de miles de compuertas en un solo paquete. Como ejemplo podemos citar las grandes matrices de memoria y los microprocesadores complejos. En virtud de su pequeño tamaño y bajo costo, los dispositivos VLSI han revolucionado la tecnología de diseño de sistemas de cómputo y confieren al diseñador la capacidad de crear estructuras que antes no resultaban económico construir.

Familias de lógica digital

Los circuitos lógicos integrados se clasifican no sólo por su complejidad o por su funcionamiento lógico, sino también por la tecnología específica de circuitos utilizada en su construcción. Llamamos a esa tecnología familia de lógica digital. Cada familia de lógica tiene su propio circuito electrónico básico sobre el que se desarrollan circuitos digitales y componentes más complejos. El circuito básico en cada tecnología es una compuerta NAND, NOR o inversora. Por lo regular, se usan los componentes electrónicos empleados en la construcción del circuito básico para dar nombre a la tecnología. Se han introducido comercialmente muchas familias lógicas de circuitos integrados digitales. Las más populares son:

TTL	lógica transistor-transistor;
ECL	lógica acoplada por emisor;
MOS	metal-óxido-semiconductor;
CMOS	metal-óxido-semiconductor complementario.

TTL es una familia lógica que ha estado en operación mucho tiempo y se le considera estándar. ECL resulta ventajoso en sistemas que deben operar a alta velocidad. MOS es apropiado para circuitos que requieren una densidad elevada de componentes, y CMOS es preferible en sistemas que requieren bajo consumo de energía. Esto último es indispensable para el diseño de VLSI, así que CMOS se ha convertido en la familia lógica dominante, mientras que el uso de TTL y ECL ha decaído. El análisis del circuito electrónico básico de una compuerta digital para cada familia de lógica se presenta en el capítulo 10.

Las características de las familias de lógica digital suelen compararse analizando el circuito de la compuerta básica de cada familia. En la sección 10-2 veremos los parámetros más importantes que se evalúan y comparan. Aquí sólo se mencionan como referencia.

El *abanico de salida* (*fan-out*) especifica el número de cargas estándar que la salida de una compuerta representativa es capaz de alimentar sin merma de su funcionamiento normal. La carga estándar por lo regular se define como la cantidad de corriente que requiere en una de sus entradas otra compuerta similar de la misma familia.

El *abanico de entrada* (*fan-in*) es el número de entradas con que cuenta la compuerta.

La *disipación de potencia* es la energía consumida por la compuerta y que la fuente de potencia debe suministrar.

El *retardo de propagación* es el tiempo medio de transición que la señal tarda al propagarse de la entrada a la salida. La velocidad de operación es inversamente proporcional al retardo de propagación.

El *margen de ruido* es el voltaje externo máximo de ruido que puede añadirse a una señal de entrada sin causar un cambio indeseable en la salida del circuito.

Diseño asistido por computadora (CAD)

El diseño de sistemas digitales con circuitos VLSI que contienen millones de transistores es una tarea imponente. En general, es imposible desarrollar y verificar sistemas tan complejos sin la ayuda de herramientas computarizadas para diseño. Las herramientas de CAD consisten en programas de software que apoyan la representación computarizada y ayudan a desarrollar hardware digital automatizando el proceso de diseño. La automatización del diseño electrónico cubre todas las fases del diseño de circuitos integrados. Un flujo de diseño típico para crear circuitos VLSI consiste en una sucesión de pasos que inicia con la introducción del diseño y cul-

mina con la generación de la base de datos que contiene la máscara fotográfica empleada para fabricar el CI. Se cuenta con diversas opciones para crear la implementación física de un circuito digital en silicio. El diseñador puede escoger entre un circuito integrado para una aplicación específica (ASIC, *application-specific integrated circuit*), un arreglo de compuertas programable en el campo (FPGA, *field-programmable gate array*), un dispositivo de lógica programable (PLD, *programmable logic device*) o un CI hecho totalmente a la medida. Cada uno de estos dispositivos viene acompañado de un surtido de herramientas de CAD que proporciona el software necesario para facilitar la fabricación en hardware de la unidad.

Algunos sistemas de CAD incluyen un programa editor para crear y modificar diagramas esquemáticos en la pantalla de una computadora. Este proceso se llama *captura de esquemas* o introducción de esquemas. Con la ayuda de menús, órdenes del teclado y el ratón, el editor de esquemas puede dibujar diagramas de circuitos digitales en la pantalla de la computadora. Es posible colocar en la pantalla componentes de una lista tomada de una biblioteca interna y luego conectarse con líneas que representan alambres. El software para captura de esquemas crea y administra una base de datos que contiene la información creada junto con el esquema. Las compuertas y bloques funcionales primitivos están asociados con modelos que permiten verificar el comportamiento y la temporización del circuito. Esta verificación se efectúa aplicando entradas al circuitto y utilizando un simulador de lógica para determinar las salidas.

Un adelanto importante en el diseño de sistemas digitales es el uso de un lenguaje de descripción de hardware (HDL). El HDL se parece a los lenguajes de programación, pero está orientado específicamente a la descripción de hardware digital. Representa diagramas de lógica y otra información digital en forma textual. Sirve para simular el sistema antes de construirlo, a fin de verificar la funcionalidad y la operación. Una aplicación importante es su software de síntesis lógica, que automatiza el diseño de sistemas digitales. El HDL ha adquirido gran importancia en años recientes y es el mejor método con que se cuenta para diseñar sistemas digitales complejos. Presentaremos el HDL en la sección 3-9 y, dada su importancia, se incluyen en todo el libro descripciones de circuitos digitales, componentes y procedimientos de diseño en HDL.

PROBLEMAS

2-1 Demuestre con tablas de verdad la validez de las identidades siguientes:

a) Teorema de DeMorgan para tres variables: $(x + y + z)' = x'y'z'$ y $(xyz)' = x' + y' + z'$

b) La ley distributiva: $x + yz = (x + y)(x + z)$

2-2 Simplifique las expresiones booleanas siguientes de modo que usen el mínimo de literales:

a) $xy + xy'$ b) $(x + y)(x + y')$

c) $xyz + x'y + xyz'$ d) $(A + B)'(A' + B')'$

2-3 Simplifique las expresiones booleanas siguientes de modo que usen el mínimo de literales:

a) $ABC + A'B + ABC'$ b) $x'yz + xz$

c) $(x + y)'(x' + y')$ d) $xy + x(wz + wz')$

e) $(BC' + A'D)(AB' + CD')$

2-4 Reduzca las siguientes expresiones booleanas al número de literales que se indica:

a) $A'C' + ABC + AC'$ a tres literales

b) $(x'y' + z)' + z + xy + wz$ a tres literales

c) $A'B(D' + C'D) + B(A + A'CD)$ a una literal

d) $(A' + C)(A' + C')(A + B + C'D)$ a cuatro literales

2-5 Obtenga el complemento de $F = x + yz$; luego demuestre que $FF' = 0$ y que $F + F' = 1$.

2-6 Obtenga el complemento de estas expresiones:

a) $xy' + x'y$ b) $(AB' + C)D' + E$

c) $(x + y' + z)(x' + z')(x + y)$

2-7 Dadas las funciones booleanas F_1 y F_2.

a) Demuestre que la función booleana $E = F_1 + F_2$ contiene la suma de los minitérminos de F_1 y F_2.

b) Demuestre que la función booleana $G = F_1F_2$ contiene únicamente los minitérminos que F_1 y F_2 tienen en común.

2-8 Muestre la tabla de verdad de la función:

$$F = xy + xy' + y'z$$

2-9 Podemos realizar operaciones lógicas con cadenas de bits considerando por separado cada par de bits correspondientes (esto se denomina operación bit por bit). Dadas dos cadenas de ocho bits $A = 10101101$ y $B = 10001110$, evalúe el resultado de ocho bits después de las operaciones lógicas siguientes: a) AND, b) OR, c) XOR, d) NOT A, e) NOT B.

2-10 Dibuje los diagramas de lógica de las expresiones booleanas siguientes:

a) $Y = A'B' + B(A + C)$ b) $Y = BC + AC'$

c) $Y = A + CD$ d) $Y = (A + B)(C' + D)$

2-11 Dada la función booleana:

$$F = xy + x'y' + y'z$$

a) impleméntela con compuertas AND, OR e inversores,

b) impleméntela con compuertas OR e inversores, y

c) impleméntela con compuertas AND e inversores.

2-12 Simplifique las funciones booleanas T_1 y T_2 al mínimo de literales.

A	*B*	*C*	T_1	T_2
0	0	0	1	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	0	1

2-13 La suma lógica de todos los minitérminos de una función booleana de n variables es 1.

a) Demuestre la afirmación anterior para $n = 3$.

b) Sugiera un procedimiento para una demostración general.

2-14 Prepare la tabla de verdad de las funciones siguientes y exprese cada función como suma de minitérminos y como producto de maxitérminos:

a) $(xy + z)(y + xz)$ b) $(A' + B)(B' + C)$

c) $y'z + wxy' + wxz' + w'x'z$

2-15 Dada la función booleana

$$F = xy'z + x'y'z + w'xy + wx'y + wxy$$

a) Prepare la tabla de verdad de la función.
b) Dibuje el diagrama de lógica empleando la expresión booleana original.
c) Simplifique la función al mínimo de literales empleando álgebra booleana.
d) Prepare la tabla de verdad de la función a partir de la expresión simplificada y demuestre que es igual a la de la parte a).
e) Dibuje el diagrama de lógica de la expresión simplificada y compare el número total de compuertas con el diagrama de la parte b).

2-16 Exprese la siguiente función como suma de minitérminos y como producto de maxitérminos.

$$F(A, B, C, D) = B'D + A'D + BD$$

2-17 Exprese el complemento de las siguientes funciones como suma de minitérminos:
a) $F(A, B, C, D) = \sum(0, 2, 6, 11, 13, 14)$ b) $F(x, y, z) = \Pi(0, 3, 6, 7)$

2-18 Convierta lo siguiente a la otra forma canónica:
a) $F(x, y, z) = \sum(1, 3, 7)$ b) $F(A, B, C, D) = \Pi(0, 1, 2, 3, 4, 6, 12)$

2-19 Convierta las expresiones siguientes a suma de productos y producto de sumas:
a) $(AB + C)(B + C'D)$ b) $x' + x(x + y')(y + z')$

2-20 Dibuje el diagrama de lógica correspondiente a las siguientes expresiones booleanas sin simplificarlas:
a) $BC' + AB + ACD$ b) $(A + B)(C + D)(A' + B + D)$
c) $(AB + A'B')(CD' + C'D)$

2-21 Demuestre que el dual del OR exclusivo es igual a su complemento.

2-22 Sustituyendo la expresión booleana equivalente de las operaciones binarias definidas en la tabla 2-8, demuestre lo siguiente:
a) La operación de inhibición no es conmutativa ni asociativa.
b) La operación de OR exclusivo es conmutativa y asociativa.

2-23 Demuestre que una compuerta NAND de lógica positiva es una compuerta NOR de lógica negativa, y viceversa.

REFERENCIAS

1. BOOLE, G. 1954. *An Investigation of the Laws of Thought*. Nueva York: Dover.
2. SHANNON, C. E. A symbolic analysis of relay and switching circuits. *Trans. AIEE* 57 (1938): 713-723.
3. HUNTINGTON, E. V. Sets of independent postulates for the algebra of logic. *Trans. Am. Math. Soc.,* **5** (1904): 288-309.
4. MANO, M. M. y C. R. KIME. 2000. *Logic and Computer Design Fundamentals*, 2a. ed. Upper Saddle River, NJ: Prentice-Hall.
5. DIETMEYER, D. L. 1988. *Logic Design of Digital Systems*, 3a. ed. Boston: Allyn Bacon.

3 Minimización en el nivel de compuertas

3-1 EL MÉTODO DEL MAPA

La complejidad de las compuertas de lógica digital que implementan una función booleana está relacionada directamente con la complejidad de la expresión algebraica a partir de la cual se implementa la función. Aunque la representación de una función como tabla de verdad es única, hay muchas formas de expresarla algebraicamente. Las expresiones booleanas se simplifican algebraicamente como se explicó en la sección 2-4, pero este procedimiento de minimización resulta poco práctico porque carece de reglas específicas que predigan cada paso sucesivo del proceso de manipulación. El método del mapa ofrece un procedimiento sencillo y directo para minimizar las funciones booleanas. Este método podría considerarse como una versión pictórica de la tabla de verdad. El método del mapa también se conoce como mapa de Karnaugh o mapa K.

El mapa es un diagrama hecho de cuadrados, cada uno de los cuales representa un minitérmino de la función. Puesto que cualquier función booleana se puede expresar como una suma de minitérminos, toda función booleana se reconocerá gráficamente en el mapa por el área delimitada por los cuadrados cuyos minitérminos están incluidos en la función. De hecho, el mapa presenta un diagrama visual de todas las maneras en que una función se puede expresar en forma estándar. Al reconocer diversos patrones, el usuario puede deducir expresiones algebraicas alternas para la misma función, y luego escoger la más simple.

Las expresiones simplificadas generadas por el mapa siempre están en una de las dos formas estándar: suma de productos o producto de sumas. Supondremos que la expresión algebraica más simple es la que tiene menos términos y el mínimo posible de literales en cada término. Esto produce un diagrama de circuito con el mínimo de compuertas y el mínimo de entradas a cada compuerta. Más adelante se verá que la expresión más simple no es única. Hay ocasiones en que es posible encontrar dos o más expresiones que satisfagan los criterios de minimización. En esos casos, cualquiera de las soluciones es satisfactoria.

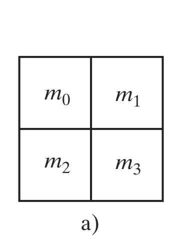
Mapa de dos variables

En la figura 3-1a) se presenta el mapa de dos variables. Hay cuatro minitérminos para dos variables; por tanto, el mapa consiste en cuatro cuadrados, uno para cada minitérmino. En b) se ha redibujado el mapa de modo que muestre la relación entre los cuadrados y las dos variables *x* y *y*. El 0 y el 1 que se marcan en cada fila y columna indican los valores de las variables. La variable *x* aparece con apóstrofo en la fila 0 y sin apóstrofo en la fila 1. De forma similar, *y* aparece con apóstrofo en la columna 0 y sin él en la columna 1.

Si marcamos los cuadrados cuyos minitérminos pertenecen a una función dada, el mapa de dos variables se convertirá en otra forma útil de representar cualquiera de las 16 funciones booleanas de dos variables. Como ejemplo, hemos mostrado la función *xy* en la figura 3-2a). Puesto que *xy* es igual a m_3, se coloca un 1 dentro del cuadrado que pertenece a m_3. Asimismo, la función $x + y$ se representa en el mapa de la figura 3-2b) con tres cuadrados marcados con unos. Esos cuadrados se obtienen de los minitérminos de la función:

$$m_1 + m_2 + m_3 = x'y + xy' + xy = x + y$$

Los tres cuadrados también podrían haberse deducido de la intersección de la variable *x* en la segunda fila y la variable *y* en la segunda columna, que encierra el área perteneciente a *x* o *y*.



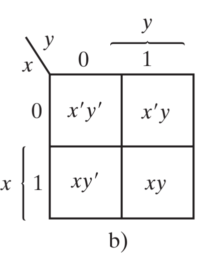
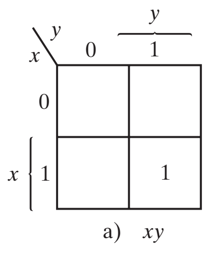


FIGURA 3-1
Mapa de dos variables



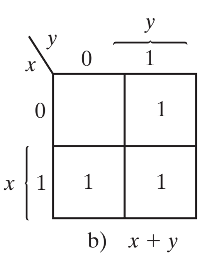


FIGURA 3-2
Representación de funciones en el mapa

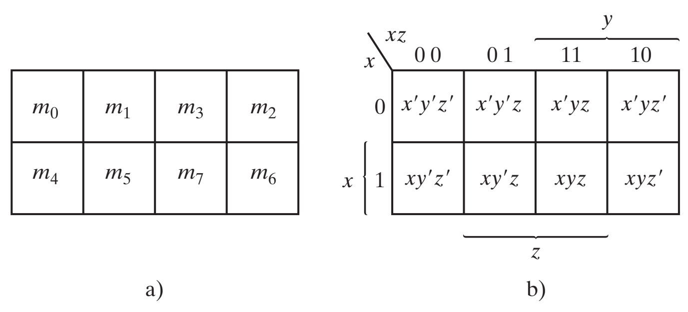


FIGURA 3-3
Mapa de tres variables

Mapa de tres variables

En la figura 3-3 se muestra un mapa de tres variables. Hay ocho minitérminos para tres variables binarias; por tanto, el mapa consta de ocho cuadrados. Advierta que los minitérminos no están acomodados en sucesión binaria, sino en una sucesión similar al código Gray (tabla 1-4). La característica de esta sucesión es que sólo un bit cambia de valor entre dos columnas adyacentes. El mapa dibujado en la parte b) se ha marcado con números en cada fila y columna que indican la relación entre los cuadrados y las tres variables. Por ejemplo, el cuadrado asignado a m_5 corresponde a la fila 1 y la columna 01. Si concatenamos estos dos números, obtendremos el número binario 101, cuyo equivalente decimal es 5. Otra forma de ver el cuadrado $m_5 = xy'z$ es considerarlo como ubicado en la fila marcada x y la columna perteneciente a $y'z$ (columna 01). Observe que hay cuatro cuadrados en los que cada variable es 1 y cuatro en los que es 0. La variable aparece sin apóstrofo en los cuatro cuadrados en los que es 1, y con apóstrofo en los que es 0. Por conveniencia, se escribe la variable con su símbolo alfabético bajo los cuatro cuadrados en los que no lleva apóstrofo.

Para entender la utilidad del mapa en la simplificación de funciones booleanas, hay que entender la propiedad básica que poseen dos cuadrados adyacentes. Cualesquier dos cuadrados adyacentes del mapa difieren en una sola variable, que tiene apóstrofo en un cuadrado y no lo tiene en el otro. Por ejemplo, m_5 y m_7 están en dos cuadrados adyacentes. La variable y tiene apóstrofo en m_5 y carece de apóstrofo en m_7, mientras que las otras dos variables son iguales en ambos cuadrados. De los postulados del álgebra booleana, se sigue que la suma de dos minitérminos en cuadrados adyacentes se simplifica a un solo término AND que consiste en sólo dos literales. Para aclarar esto, consideremos la suma de dos cuadrados adyacentes como m_5 y m_7:

$$m_5 + m_7 = xy'z + xyz = xz(y' + y) = xz$$

Aquí, los dos cuadrados difieren en la variable y, que se elimina al formarse la suma de los dos minitérminos. Así, cualesquier dos minitérminos en cuadrados adyacentes a los que se aplique un OR permitirán eliminar la variable diferente. Los ejemplos que siguen explican el procedimiento para minimizar una función booleana con un mapa.

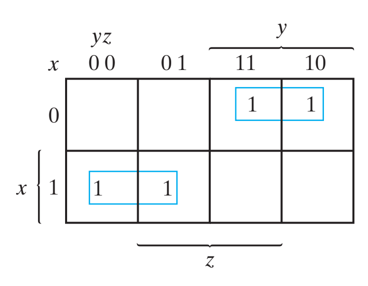


FIGURA 3-4
Mapa para el ejemplo 3-1; $F(x, y, z) = \sum(2, 3, 4, 5) = x'y + xy'$

EJEMPLO 3-1

Simplifique la función booleana

$$F(x, y, z) = \sum(2, 3, 4, 5)$$

Primero, marcamos con un 1 cada uno de los minitérminos que representan a la función. Esto se indica en la figura 3-4, donde se han marcado con 1 los cuadrados correspondientes a los minitérminos 010, 011, 100 y 101. El siguiente paso es encontrar los posibles cuadrados adyacentes, que se indican en el mapa con dos rectángulos, cada uno de los cuales encierra dos unos. El rectángulo de arriba a la derecha representa el área delimitada por $x'y$. Esto se determina observando que el área de dos cuadrados está en la fila 0, que corresponde a x', y las últimas dos columnas, que corresponden a y. De forma similar, el rectángulo inferior izquierdo representa el término de producto xy'. (La segunda fila representa a x y las dos columnas de la izquierda representan a y'.) La suma lógica de estos dos términos producto da la expresión simplificada:

$$F = x'y + xy'$$

Hay casos en los que se considera que dos cuadrados del mapa están adyacentes, aunque no se estén tocando. En la figura 3-3, m_0 es adyacente a m_2 y m_4 adyacente a m_6 porque los minitérminos difieren en una sola variable. Esto se puede verificar fácilmente con álgebra:

$$m_0 + m_2 = x'y'z' + x'yz' = x'z'(y' + y) = x'z'$$
$$m_4 + m_6 = xy'z' + xyz' = xz' + (y' + y) = xz'$$

Por tanto, deberemos modificar la definición de cuadrados adyacentes de modo que incluya este caso y otros similares. Esto se hace considerando que el mapa está dibujado en una superficie cuyos bordes derecho e izquierdo están en contacto para formar cuadrados adyacentes.

EJEMPLO 3-2

Simplifique la función booleana

$$F(x, y, z) = \sum(3, 4, 6, 7)$$

El mapa para esta función se presenta en la figura 3-5. Hay cuatro cuadrados marcados con unos, uno para cada minitérmino de la función. En la tercera columna se combinan dos cuadrados

x \ yz	00	01	11	10
0			1	
1	1		1	1

FIGURA 3-5
Mapa para el ejemplo 3-2; $F(x, y, z) = \sum(3, 4, 6, 7) = yz + xz'$

adyacentes para dar un término de dos literales, yz. Los otros dos cuadrados que incluyen unos también son adyacentes según la nueva definición, por lo que en el diagrama sus valores se encierran con medios rectángulos. Estos dos cuadrados combinados dan el término de dos literales xz'. La función simplificada es

$$F = yz + xz'$$

Considere ahora cualquier combinación de cuatro cuadrados adyacentes en el mapa de tres variables. Cualquier combinación así representa la suma lógica de cuatro minitérminos y da como resultado una expresión con una sola literal. Por ejemplo, la suma lógica de los cuatro minitérminos adyacentes 0, 2, 4 y 6 se reduce a un término de una sola literal, z':

$$\begin{aligned} m_0 + m_2 + m_4 + m_6 &= x'y'z' + x'yz' + xy'z' + xyz' \\ &= x'z'(y' + y) + xz'(y' + y) \\ &= x'z' + xz' = z'(x' + x) = z' \end{aligned}$$

El número de cuadrados adyacentes que es posible combinar siempre debe ser una potencia de 2, como 1, 2, 4 y 8. Al aumentar el número de cuadrados adyacentes que se combinan, se reduce el número de literales del término producto obtenido.

Un cuadrado representa un minitérmino, lo que da un término con tres literales.

Dos cuadrados adyacentes representan un término de dos literales.

Cuatro cuadrados adyacentes representan un término de una sola literal.

Ocho cuadrados adyacentes abarcan todo el mapa y producen una función que siempre es igual a 1.

EJEMPLO 3-3

Simplifique la función booleana

$$F(x, y, z) = \sum(0, 2, 4, 5, 6)$$

El mapa de F se muestra en la figura 3-6. Primero, combinamos los cuatro cuadrados adyacentes de la primera y la última columnas para obtener el término de una sola literal z'. El cuadrado restante, que representa el minitérmino 5, se combina con un cuadrado adyacente que ya se usó una vez. Esto no sólo es permisible, sino hasta deseable, porque los dos cuadrados adya-

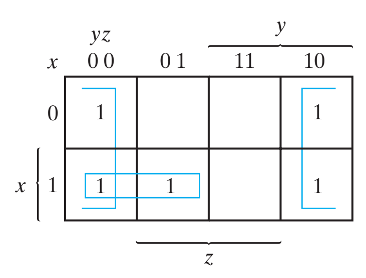


FIGURA 3-6
Mapa para el ejemplo 3-3; $F(x, y, z) = \sum(0, 2, 4, 5, 6) = z' + xy'$

centes dan el término de dos literales xy', mientras que el cuadrado solo representa el minitérmino de tres literales $xy'z$. la función simplificada es

$$F = z' + xy'$$

■

Si una función no está expresada como suma de minitérminos, podemos usar el mapa para obtener los minitérminos de la función y luego simplificar la función a una expresión con el mínimo de términos. Hay que asegurarse de que la expresión algebraica esté en forma de suma de productos. Cada término de producto se puede marcar en el mapa en uno, dos o más cuadrados. Luego, los minitérminos de la función se leen directamente del mapa.

EJEMPLO 3-4

Dada la función booleana

$$F = A'C + A'B + AB'C + BC$$

a) exprésela como suma de minitérminos

b) y luego halle la expresión mínima de suma de productos.

Tres términos de producto de la expresión tienen dos literales y se representan en un mapa de tres variables con dos cuadrados cada uno. Los dos cuadrados correspondientes al primer término, $A'C$, se encuentran en la figura 3-7 en la intersección de A' (primera fila) y C (dos columnas de en medio), que da los cuadrados 001 y 011. Observe que, al marcar los cuadrados

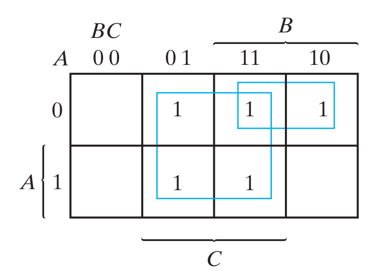


FIGURA 3-7
Mapa para el ejemplo 3-4; $A'C + A'B + AB'C + BC = C + A'B$

con 1, podríamos hallar ahí un 1 colocado por un término anterior. Esto sucede con el segundo término, $A'B$, que tiene unos en los cuadrados 011 y 010. Sin embargo, el cuadrado 011 también corresponde al primer término, $A'C$, así que simplemente dejamos el 1 que ya está ahí. Continuando de la misma manera, determinamos que el término $AB'C$ va en el cuadrado 101, que corresponde al minitérmino 5, y que el término BC tiene dos unos en los cuadrados 011 y 111. La función tiene un total de cinco minitérminos, como puede verse por los cinco unos en el mapa de la figura 3-7. Los minitérminos se leen directamente del mapa, y son 1, 2, 3, 5 y 7. La función se expresa en forma de suma de minitérminos:

$$F(A, B, C) = \sum(1, 2, 3, 5, 7)$$

La expresión de suma de productos dada originalmente tiene demasiados términos; puede simplificarse, como se muestra en el mapa, a una expresión de sólo dos términos:

$$F = C + A'B$$

■

3-2 MAPA DE CUATRO VARIABLES

El mapa para las funciones booleanas de cuatro variables se ilustra en la figura 3-8. En a) se presentan los 16 minitérminos y los cuadrados asignados a cada uno. En b) se ha redibujado el mapa de modo que muestre su relación con las cuatro variables. Las filas y columnas se numeran en orden según el código Gray, de modo que sólo un dígito cambie de valor entre dos filas o columnas adyacentes. El minitérmino correspondiente a cada cuadrado se obtiene de la concatenación del número de fila con el número de columna. Por ejemplo, los números de la tercera fila (11) y la segunda columna (01) dan, al concatenarse, el número binario 1101, que es el equivalente binario del 13 decimal. Así, el cuadrado de la tercera fila y la segunda columna representa al minitérmino m_{13}.

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}

a)

$wx \backslash yz$	00	01 (z)	11 (y, z)	10 (y)
00	$w'x'y'z'$	$w'x'y'z$	$w'x'yz$	$w'x'yz'$
01 (x)	$w'xy'z'$	$w'xy'z$	$w'xyz$	$w'xyz'$
11 (w, x)	$wxy'z'$	$wxy'z$	$wxyz$	$wxyz'$
10 (w)	$wx'y'z'$	$wx'y'z$	$wx'yz$	$wx'yz'$

b)

FIGURA 3-8
Mapa de cuatro variables

La minimización por mapa de funciones booleanas de cuatro variables es similar al método que se emplea para minimizar funciones de tres variables. Definimos los cuadrados adyacentes como cuadrados que están juntos. Además, consideramos que el mapa está en una superficie cuyos bordes superior e inferior, y derecho e izquierdo, están en contacto para formar cuadrados adyacentes. Por ejemplo, m_0 y m_2 forman cuadrados adyacentes, lo mismo que m_3 y m_{11}. Es fácil determinar la combinación de cuadrados adyacentes que es útil para el proceso de simplificación, por inspección del mapa de cuatro variables:

Un cuadrado representa un minitérmino, lo que da un término con cuatro literales.

Dos cuadrados adyacentes representan un término de tres literales.

Cuatro cuadrados adyacentes representan un término de dos literales.

Ocho cuadrados adyacentes representan un término de una sola literal.

Dieciséis cuadrados adyacentes representan la función igual a 1.

Ninguna otra combinación de cuadrados puede simplificar la función. Los dos ejemplos siguientes ilustran el uso del procedimiento para simplificar funciones booleanas de cuatro variables.

EJEMPLO 3-5

Simplifique la función booleana

$$F(w, x, y, z) = \sum(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$$

Puesto que la función tiene cuatro variables, hay que usar un mapa de cuatro variables. Los minitérminos indicados en la suma se han marcado con 1 en el mapa de la figura 3-9. Es posible combinar ocho cuadrados adyacentes marcados con 1 para formar el término de una literal y'. Los tres unos restantes de la derecha no pueden combinarse para dar un término simplificado. Se deberán combinar como dos o cuatro cuadrados adyacentes. Cuanto mayor sea el número

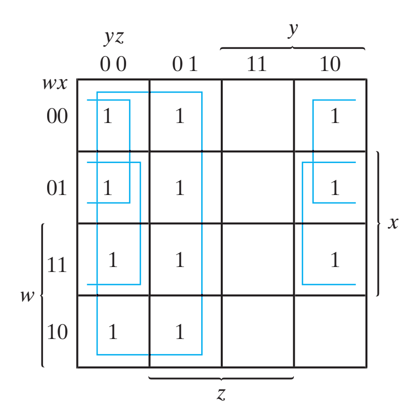


FIGURA 3-9
Mapa para el ejemplo 3-5; $F(w, x, y, z) = \sum(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$ $= y' + w'z' + xz'$

de cuadrados combinados, menos literales tendrá el término correspondiente. En este ejemplo, los dos unos de arriba a la derecha se combinan con los dos unos de arriba a la izquierda para dar el término $w'z'$. Cabe señalar que está permitido usar el mismo cuadrado más de una vez. Ahora nos queda un cuadrado marcado con 1 en la tercera fila y la cuarta columna (cuadrado 1110). En vez de tomar este cuadrado solo (lo que daría un término con cuatro literales), lo combinamos con cuadrados que ya usamos antes para formar un área de cuatro cuadrados adyacentes. Estos cuadrados ocupan la intersección de las dos filas de en medio y las dos columnas de los extremos, y dan el término xz'. La función simplificada es

$$F = y' + w'z' + xz'$$

■

EJEMPLO 3-6

Simplificar la función booleana

$$F = A'B'C' + B'CD' + A'BCD' + AB'C'$$

El área del mapa cubierta por esta función consiste en los cuadrados marcados con 1 en la figura 3-10. Esta función tiene cuatro variables y, en la forma en que está expresada, consta de tres términos de tres literales cada uno y un término de cuatro literales. Cada término de tres literales se representa en el mapa con dos cuadrados. Por ejemplo, $A'B'C'$ se representa en los cuadrados 0000 y 0001. La función se puede simplificar en el mapa tomando los unos de las cuatro esquinas para dar el término $B'D'$. Esto es posible porque los cuatro cuadrados están adyacentes si el mapa se dibuja en una superficie cuyos bordes superior e inferior, y derecho e izquierdo, están en contacto. Los dos unos de la izquierda en la fila superior se combinan con los dos unos de la fila inferior para dar el término $B'C'$. El uno restante se puede combinar en un área de dos cuadrados para dar el término $A'CD'$. La función simplificada es

$$F = B'D' + B'C' + A'CD'$$

■

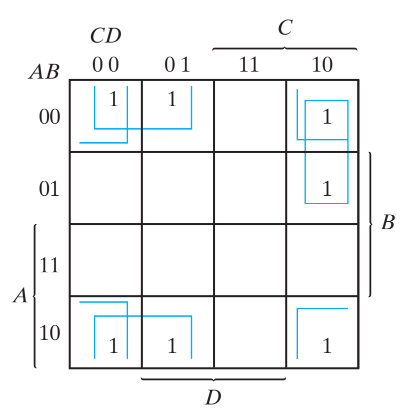


FIGURA 3-10
Mapa para el ejemplo 3-6; $A'B'C' + B'CD' + A'BCD' + AB'C' = B'D' + B'C' + A'CD'$

Implicantes primos

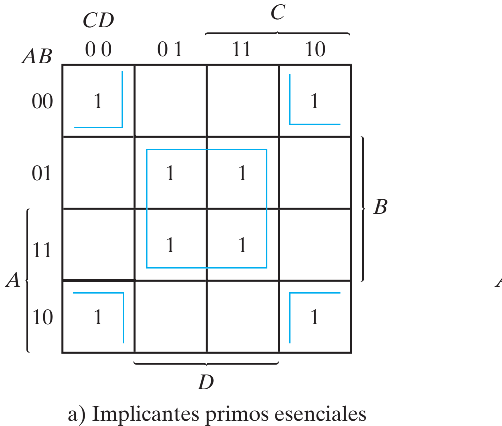
Al escoger cuadrados adyacentes en un mapa, debemos asegurarnos de cubrir todos los minitérminos de la función al combinar los cuadrados. Al mismo tiempo, es necesario minimizar el número de términos de la expresión y evitar términos redundantes cuyos minitérminos ya estén cubiertos por otros términos. Ocasionalmente, habrá dos o más expresiones que satisfacen los criterios de simplificación. El procedimiento para combinar cuadrados en el mapa podría hacerse más sistemático si entendemos el significado de los términos denominados implicante primo e implicante primo esencial. Un *implicante primo* es un término de producto que se obtiene combinando el número máximo posible de cuadrados adyacentes en el mapa. Si un minitérmino de un cuadrado está cubierto por sólo un implicante primo, decimos que ese implicante primo es *esencial*.

Podemos obtener los implicantes primos de una función a partir de un mapa combinando todos los números máximos posibles de cuadrados. Esto implica que un solo 1 en un mapa representa un implicante primo si no está adyacente a ningún otro 1. Dos unos adyacentes forman un implicante primo si no están dentro de un grupo de cuatro cuadrados adyacentes. Cuatro unos adyacentes forman un implicante primo si no están dentro de un grupo de ocho cuadrados adyacentes, y así sucesivamente. Los implicantes primos esenciales se encuentran examinando cada uno de los cuadrados marcados con 1 y tomando nota del número de implicantes primos que lo cubren. El implicante primo es esencial si es el único que cubre al minitérmino.

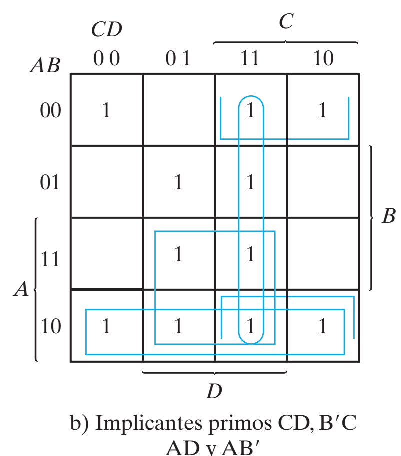
Considere la siguiente función booleana de cuatro variables:

$$F(A, B, C, D) = \sum(0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)$$

Los minitérminos de la función se han marcado con 1 en los mapas de la figura 3-11. La parte a) de la figura muestra dos implicantes primos esenciales. Un término es esencial porque sólo hay una forma de incluir el minitérmino m_0 en cuatro cuadrados adyacentes. Estos cuatro cuadrados definen al término $B'D'$. Asimismo, sólo hay una forma de combinar el minitérmi-



a) Implicantes primos esenciales
BD y B′D′



b) Implicantes primos CD, B′C
AD y AB′

FIGURA 3-11
Simplificación empleando implicantes primos

no m_5 con cuatro cuadrados adyacentes, y esto da el segundo término, *BD*. Los dos implicantes primos esenciales cubren ocho minitérminos. Ahora hay que considerar los tres minitérminos restantes, m_3, m_9 y m_{11}.

La figura 3-11b) muestra todas las formas posibles en que se cubren los tres minitérminos con implicantes primos. El minitérmino m_3 puede cubrirse con el implicante primo *CD* o con el $B'C$. El minitérmino m_9 puede cubrirse con *AD* o con AB'. El minitérmino m_{11} se cubre con cualquiera de los cuatro implicantes primos. La expresión simplificada se obtiene de la suma lógica de los dos implicantes primos esenciales y de cualesquier dos implicantes primos que cubran los minitérminos m_3, m_9 y m_{11}. Hay cuatro posibles formas de expresar la función con cuatro términos de producto de dos literales cada uno:

$$
\begin{aligned}
F &= BD + B'D' + CD + AD \\
&= BD + B'D' + CD + AB' \\
&= BD + B'D' + B'C + AD \\
&= BD + B'D' + B'C + AB'
\end{aligned}
$$

El ejemplo anterior demuestra que la identificación de los implicantes primos en el mapa ayuda a determinar las alternativas con que se cuenta para obtener una expresión simplificada.

El procedimiento para obtener del mapa la expresión simplificada requiere identificar primero todos los implicantes primos esenciales. La expresión simplificada se obtiene de la suma lógica de todos los implicantes primos esenciales y los demás implicantes primos necesarios para cubrir los minitérminos restantes que no estén cubiertos por los implicantes primos esenciales. Ocasionalmente, habrá más de una manera de combinar cuadrados, y cada combinación podría dar pie a una expresión igualmente simplificada.

3-3 MAPA DE CINCO VARIABLES

El uso de mapas para más de cuatro variables no es tan sencillo. Un mapa de cinco variables necesita 32 cuadrados, y uno de seis variables, 64 cuadrados. Cuando hay muchas variables, el número de cuadrados aumenta en forma considerable y la geometría para combinar cuadrados adyacentes se complica progresivamente.

El mapa de cinco variables se muestra en la figura 3-12. Consta de dos mapas de cuatro variables con las variables *A*, *B*, *C*, *D* y *E*. La variable *A* distingue a los dos mapas, como se indica en la parte superior del diagrama. El mapa de cuatro variables de la izquierda representa los 16 cuadrados en los que $A = 0$; el otro representa los cuadrados en los que $A = 1$. Los minitérminos 0 a 15 corresponden a $A = 0$ y los minitérminos 16 a 31 corresponden a $A = 1$. Cada mapa de cuatro variables conserva las adyacencias que definimos antes cuando se le considera aparte. Además, cada cuadrado del mapa $A = 0$ es adyacente al cuadrado correspondiente del mapa $A = 1$. Por ejemplo, el minitérmino 4 es adyacente al minitérmino 20, y el minitérmino 15, al 31. La mejor forma de visualizar esta nueva regla de adyacencia es imaginar que los dos medios mapas están uno encima del otro. Cualesquier dos cuadrados que queden uno encima del otro se considerarán adyacentes.

Siguiendo el procedimiento empleado con el mapa de cinco variables, es posible construir un mapa de seis variables con cuatro mapas de cuatro variables, para obtener los 64 cuadrados necesarios. Los mapas con seis o más variables requieren demasiados cuadrados y su uso

$A = 0$

BC \ DE	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

$A = 1$

BC \ DE	00	01	11	10
00	16	17	19	18
01	20	21	23	22
11	28	29	31	30
10	24	25	27	26

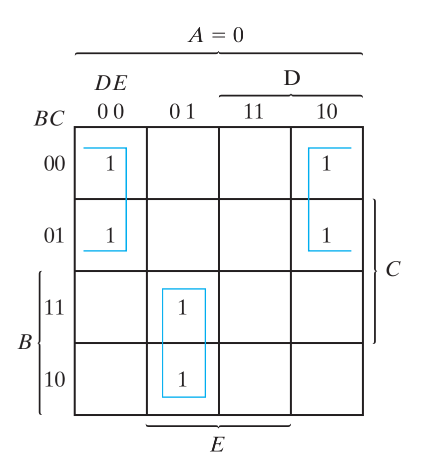
FIGURA 3-12
Mapa de cinco variables

resulta impráctico. La alternativa es utilizar programas de computadora escritos específicamente para facilitar la simplificación de funciones booleanas que tienen un gran número de variables.

Por inspección, y tomando en cuenta la nueva definición de cuadrados adyacentes, es posible demostrar que cualesquier 2^k cuadrados adyacentes, para $k = (0, 1, 2, \ldots, n)$, en un mapa de n variables, representan un área que produce un término de $n - k$ literales. Para que esta afirmación tenga sentido, n deberá ser mayor que k. Cuando $n = k$, toda el área del mapa se combina para dar la función de identidad. La tabla 3-1 muestra la relación entre el número de cuadrados adyacentes y el número de literales en el término. Por ejemplo, ocho cuadrados adyacentes combinan un área del mapa de cinco variables para dar un término de dos literales.

Tabla 3-1
Relación entre el número de cuadrados adyacentes y el número de literales en el término

	Número de cuadrados adyacentes	**Número de literales del término en un mapa de *n* variables**			
K	2^k	$n = 2$	$n = 3$	$n = 4$	$n = 5$
0	1	2	3	4	5
1	2	1	2	3	4
2	4	0	1	2	3
3	8		0	1	2
4	16			0	1
5	32				0



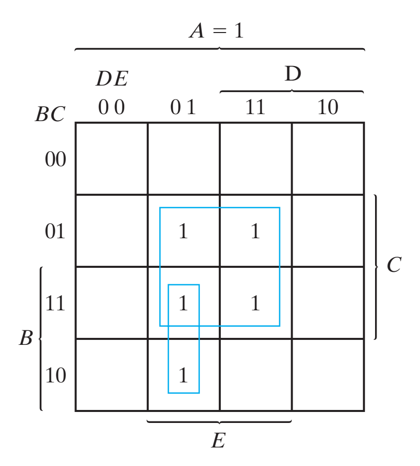


FIGURA 3-13
Mapa para el ejemplo 3-7; $F = A'B'E' + BD'E + ACE$

EJEMPLO 3-7

Simplifique la función booleana

$$F(A, B, C, D, E) = (0, 2, 4, 6, 9, 13, 21, 23, 25, 29, 31)$$

El mapa de cinco variables para esta función se muestra en la figura 3-13. Hay seis minitérminos, del 0 al 15, que pertenecen a la parte del mapa en la que $A = 0$. Los otros cinco minitérminos pertenecen a $A = 1$. Cuatro cuadrados adyacentes del mapa $A = 0$ se combinan para dar el término de tres literales $A'B'E'$. Advierta que es necesario incluir A' en el término porque todos los cuadrados están asociados a $A = 0$. Los dos cuadrados de la columna 01 y las dos últimas filas son comunes a ambas partes del mapa; por tanto, constituyen cuatro cuadrados adyacentes y dan el término de tres literales $BD'E$. La variable A no se incluye aquí porque los cuadrados adyacentes pertenecen tanto a $A = 0$ como a $A = 1$. El término ACE se obtiene de los cuatro cuadrados adyacentes que están totalmente dentro del mapa $A = 1$. La función simplificada es la suma lógica de los tres términos:

$$F = A'B'E' + BD'E + ACE$$

■

3-4 SIMPLIFICACIÓN DE PRODUCTO DE SUMAS

Las funciones booleanas simplificadas que dedujimos del mapa en todos los ejemplos anteriores se expresaron en la forma de suma de productos. Con una modificación menor, se obtiene la forma de producto de sumas.

El procedimiento para obtener una función minimizada en forma de producto de sumas es consecuencia de las propiedades básicas de las funciones booleanas. Los unos que se colocan en los cuadrados del mapa representan los minitérminos de la función. Los minitérminos no in-

cluidos en la función denotan el complemento de la función. Entonces, el complemento de una función está representado en el mapa por los cuadrados que no se han marcado con 1. Si marcamos los cuadrados vacíos con 0 y los combinamos en cuadrados adyacentes válidos, obtendremos una expresión simplificada del complemento de la función, es decir, de F'. El complemento de F' nos dará otra vez la función F. Por el teorema generalizado de DeMorgan, la función así obtenida estará automáticamente en forma de producto de sumas. La mejor forma de explicar esto es con un ejemplo.

EJEMPLO 3-8

Simplifique la siguiente función booleana en forma de **a)** suma de productos y **b)** producto de sumas:

$$F(A, B, C, D) = \sum(0, 1, 2, 5, 8, 9, 10)$$

Los unos marcados en el mapa de la figura 3-14 representan todos los minitérminos de la función. Los cuadrados marcados con 0 representan los minitérminos no incluidos en F y, por tanto, denotan al complemento de F. Si combinamos los cuadrados que tienen 1, obtendremos la función simplificada en forma de suma de productos:

a) $F = B'D' + B'C' + A'C'D$

Si combinamos los cuadrados marcados con 0, como se indica en el diagrama, obtendremos la función complementada simplificada:

$$F' = AB + CD + BD'$$

Al aplicar el teorema de DeMorgan (obteniendo el dual y complementando cada literal como se describe en la sección 2-4) se obtiene la función simplificada en forma de producto de sumas:

b) $F = (A' + B')(C' + D')(B' + D)$

■

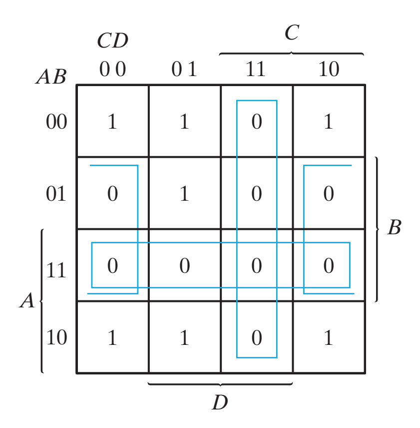
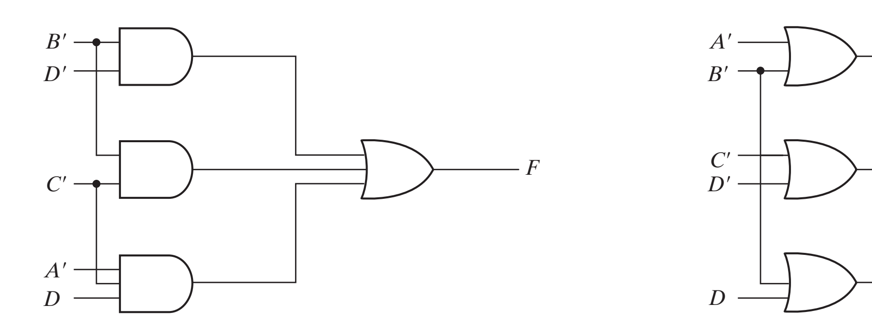


FIGURA 3-14
Mapa para el ejemplo 3-8; $F(A, B, C, D) = \sum(0, 1, 2, 5, 8, 9, 10)$
$= B'D' + B'C' + A'C'D = (A' + B')(C' + D')(B' + D)$



a) $F = B'D' + B'C' + A'C'D$

b) $F = (A' + B')(C' + D')(B' + D)$

FIGURA 3-15
Implementación con compuertas de la función del ejemplo 3-8

La implementación de las expresiones simplificadas obtenidas en el ejemplo 3-8 se muestra en la figura 3-15. La expresión de suma de productos se implementa en a) con un grupo de compuertas AND, una para cada término AND. Las salidas de las compuertas AND se conectan a las entradas de una sola compuerta OR. En b) se ha implementado la misma función en forma de producto de sumas con un grupo de compuertas OR, una por cada término OR. Las salidas de las compuertas OR se conectan a las entradas de una sola compuerta AND. En cada caso, suponemos que contamos directamente con las variables de entrada complementadas, por lo que no se necesitan inversores. El patrón de configuración establecido en la figura 3-15 es la forma general de implementar cualquier función booleana expresada en una de las formas estándar. Si está en suma de productos, se conectan compuertas AND a una sola compuerta OR; si está en producto de sumas, se conectan compuertas OR a una sola compuerta AND. Cualquiera de las configuraciones forma dos niveles de compuertas, por lo que se afirma que la implementación de una función en forma estándar es una implementación de dos niveles.

El ejemplo 3-8 ilustró el procedimiento para obtener la simplificación de producto de sumas cuando la función se expresa originalmente en la forma canónica de suma de minitérminos. El mismo procedimiento es válido cuando la función se expresa originalmente en la forma

Tabla 3-2
Tabla de verdad de la función F

x	*y*	*z*	*F*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

x \ yz	00	01	11	10
0	0	1	1	0
1	1	0	0	1

FIGURA 3-16
Mapa de la función de la tabla 3-2

canónica de producto de maxitérminos. Consideremos, por ejemplo, la tabla de verdad que define la función F en la tabla 3-2. En suma de minitérminos, esta función se expresa así:

$$F(x, y, z) = \sum(1, 3, 4, 6)$$

En producto de maxitérminos, se expresa así:

$$F(x, y, z) = \Pi\,(0, 2, 5, 7)$$

Dicho de otro modo, los unos de la función representan a los minitérminos, y los ceros, a los maxitérminos. El mapa de esta función se presenta en la figura 3-16. Podemos comenzar a simplificar esta función marcando con 1 los cuadrados correspondientes a cada minitérmino para el cual la función da 1. Los cuadrados restantes se marcan con 0. Por otra parte, si lo que se da originalmente es el producto de maxitérminos, podemos comenzar por colocar ceros en los cuadrados indicados por la función; luego marcaremos con 1 los cuadrados restantes. Una vez marcados todos los cuadrados, es posible simplificar la función en cualquiera de las formas estándar. Si queremos la suma de productos, combinaremos los unos para obtener

$$F = x'z + xz'$$

Si queremos el producto de sumas, combinaremos los ceros para obtener la función complementada simplificada

$$F' = xz + x'z'$$

lo que demuestra que la función OR exclusivo es el complemento de la función de equivalencia (sección 2-6). Al calcular el complemento de F', se obtiene la función simplificada en forma de producto de sumas:

$$F = (x' + z')(x + z)$$

Para introducir en el mapa una función expresada como producto de sumas, se calcula el complemento de la función, el cual indicará los cuadrados que deben marcarse con 0. Por ejemplo, la función

$$F = (A' + B' + C')(B + D)$$

se puede introducir en el mapa obteniendo primero su complemento,

$$F' = ABC + B'D'$$

y marcando después con 0 los cuadrados que representan a los minitérminos de F'. Los cuadrados restantes se marcarán con 1.

3-5 CONDICIONES DE INDIFERENCIA

La suma lógica de los minitérminos asociados con una función booleana especifica las condiciones en que la función da 1. La función da 0 para el resto de los minitérminos. Esto supone que todas las combinaciones de valores de las variables de la función son válidas. En la práctica, hay algunas aplicaciones en las que la función no está especificada para ciertas combinaciones de las variables. Por ejemplo, el código binario de cuatro bits para los dígitos decimales tiene seis combinaciones que no se usan y que, por tanto, se consideran no especificadas. Las funciones con salidas no especificadas para ciertas combinaciones de entradas se llaman funciones incompletamente especificadas. En casi todas las aplicaciones, es irrelevante el valor que asuma la función para los minitérminos no especificados. Por ello, se acostumbra llamar condiciones de indiferencia (*don't care*, en inglés) a los minitérminos no especificados de una función. Conviene usar estas condiciones de indiferencia en el mapa para simplificar aún más la expresión booleana.

Debe quedar claro que un minitérmino indiferente es una combinación de variables cuyo valor lógico no está especificado. No podemos marcarlo con 1 en el mapa porque ello requeriría que la función siempre dé 1 para esa combinación. Por lo mismo, si marcamos con 0 su cuadrado en el mapa, estaremos exigiendo que la función sea 0. Para distinguir la condición de indiferencia, usamos una *X* en lugar de unos y ceros. Así, una *X* en un cuadrado del mapa indica que no nos importa si se asigna el valor de 0 o de 1 a *F* para el minitérmino en cuestión.

Al escoger cuadrados adyacentes para simplificar la función, podemos suponer que los minitérminos indiferentes son 0 o 1, lo que más nos convenga. Al simplificar la función, podemos optar por incluir cada minitérmino indiferente con los unos o con los ceros, dependiendo de qué combinación produzca la expresión más simple.

EJEMPLO 3-9

Simplifique la función booleana

$$F(w, x, y, z) = \Sigma(1, 3, 7, 11, 15)$$

que tiene las condiciones de indiferencia

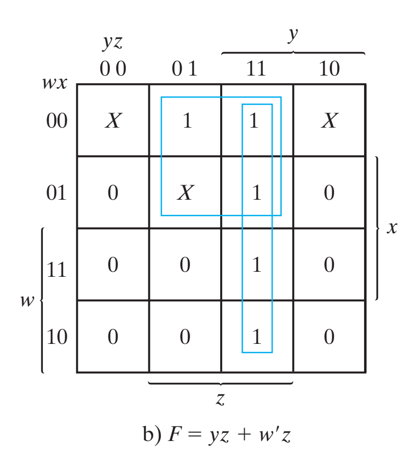
$$d(w, x, y, z) = \Sigma(0, 2, 5)$$

Los minitérminos de *F* son las combinaciones de variables que hacen que la función dé 1. Los minitérminos de *d* son los minitérminos indiferentes a los que podría asignarse 0 o bien 1. La simplificación del mapa se muestra en la figura 3-17. Los minitérminos de *F* se marcan con 1, los de *d* se marcan con *X* y los cuadrados restantes se marcan con 0. Para obtener la expresión simplificada en forma de suma de productos, deberemos incluir los cinco unos del mapa, pero podríamos incluir o no cualquiera de las *X*, dependiendo de qué tanto ello simplifique la función. El término yz cubre los cuatro minitérminos de la tercera columna. El minitérmino restante, m_1, se puede combinar con el minitérmino m_3 para dar el término de tres literales $w'x'z$. Pero si incluimos una o dos *X* adyacentes podremos combinar cuatro cuadrados adyacentes para obtener un término de dos literales. En la parte a) del diagrama se han incluido los minitérminos indiferentes 0 y 2 junto con los unos, para dar la función simplificada

$$F = yz + w'x'$$

wx \ yz	00	01	11	10
00	X	1	1	X
01	0	X	1	0
11	0	0	1	0
10	0	0	1	0

a) $F = yz + w'x'$



b) $F = yz + w'z$

FIGURA 3-17
Ejemplo con condiciones de indiferencia

En la parte b), se ha incluido el minitérmino 5 junto con los unos, y ahora la función simplificada es

$$F = yz + w'z$$

Cualquiera de las dos expresiones anteriores satisface las condiciones planteadas para el ejemplo.

■

El ejemplo anterior ilustró cómo los minitérminos indiferentes se marcan inicialmente con *X* en el mapa y se consideran como 0 o como 1. La decisión de tomar cada uno como 0 o como 1 dependerá de cómo ello simplifique la función incompletamente especificada. Una vez tomada la decisión, la función simplificada obtenida consistirá en una suma de minitérminos que incluye los minitérminos que inicialmente no estaban especificados y que ahora se ha decidido incluir con los unos. Consideremos las dos expresiones simplificadas que se obtienen en el ejemplo 3-9:

$$F(w, x, y, z) = yz + w'x' = \Sigma(0, 1, 2, 3, 7, 11, 15)$$
$$F(w, x, y, z) = yz + w'z = \Sigma(1, 3, 5, 7, 11, 15)$$

Ambas expresiones incluyen a los minitérminos 1, 3, 7, 11 y 15, que hacen a la función *F* igual a 1. Los minitérminos indiferentes 0, 2 y 5 se tratan de diferente manera en cada expresión. La primera expresión incluye a los minitérminos 0 y 2 junto con los unos y deja al minitérmino 5 con los ceros. La segunda expresión incluye al minitérmino 5 con los unos y deja a los minitérminos 0 y 2 con los ceros. Las dos funciones son algebraicamente distintas. Ambas cubren los minitérminos especificados de la función, pero cada una cubre diferentes minitérminos indiferentes. En lo que a la función incompletamente especificada concierne, cualquiera de las dos expresiones es aceptable porque la única diferencia radica en el valor de *F* para los minitérminos indiferentes.

También es posible obtener una expresión simplificada de producto de sumas para la función de la figura 3-17. En este caso, la única forma de combinar los ceros es incluyendo los minitérminos indiferentes 0 y 2 con los ceros, para dar una función complementada simplificada:

$$F' = z' + wy'$$

Al calcular el complemento de F' obtenemos la expresión simplificada como producto de sumas:

$$F(w, x, y, z) = z(w' + y) = \sum(1, 3, 5, 7, 11, 15)$$

En este caso, hemos incluido los minitérminos 0 y 2 con los ceros, y el minitérmino 5 con los unos.

3-6 IMPLEMENTACIÓN CON NAND Y NOR

Muchos circuitos digitales se construyen con compuertas NAND y NOR en lugar de con compuertas AND y OR. Las primeras son más fáciles de fabricar con componentes electrónicos y son las compuertas básicas empleadas en todas las familias de lógica de CI digitales. En virtud del destacado papel que las compuertas NAND y NOR desempeñan en el diseño de circuitos digitales, se han desarrollado reglas y procedimientos para convertir funciones booleanas expresadas en términos de AND, OR y NOT en diagramas lógicos NAND y NOR equivalentes.

Circuitos NAND

Se dice que la compuerta NAND es una compuerta universal porque cualquier sistema digital puede implementarse con ella. Para demostrar que cualquier función booleana se puede implementar con compuertas NAND, basta con demostrar que las operaciones lógicas AND, OR y complemento se pueden obtener exclusivamente con compuertas NAND. Esto se aprecia en la figura 3-18. La operación complemento se obtiene con una compuerta NAND de una sola entrada que se comporta exactamente como un inversor. La operación AND requiere dos compuertas NAND. La primera produce la operación NAND y la segunda invierte el sentido lógico de la señal. La operación OR se logra con una compuerta NAND que lleva inversores en cada entrada.

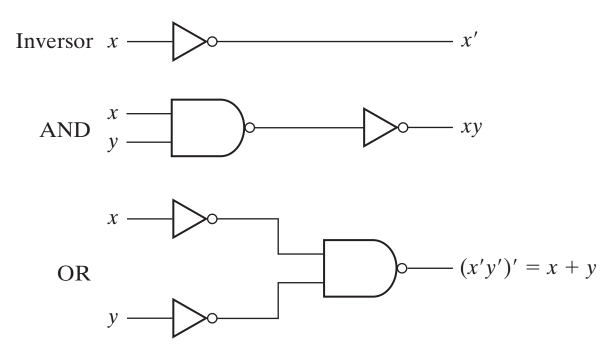


FIGURA 3-18
Operaciones lógicas con compuertas NAND

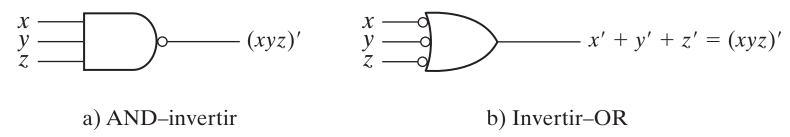


FIGURA 3-19
Dos símbolos gráficos para la compuerta NAND

Una forma conveniente de implementar una función booleana con compuertas NAND consiste en obtener la función booleana simplificada en términos de operadores booleanos y luego convertir la función a lógica NAND. La conversión de una expresión algebraica de AND, OR y complemento a NAND se efectúa aplicando sencillas técnicas de manipulación de circuitos que convierten los diagramas AND-OR en diagramas NAND.

Para facilitar la conversión a lógica NAND, es conveniente definir un símbolo gráfico alterno para la compuerta. En la figura 3-19 se presentan dos símbolos gráficos equivalentes para la compuerta NAND. El símbolo AND-invertir ya se definió antes y consiste en un símbolo gráfico AND seguido de un pequeño indicador circular de negación llamado burbuja. Como alternativa, podemos representar una compuerta NAND con un símbolo gráfico OR precedido por una burbuja en cada entrada. El símbolo invertir-OR para la compuerta NAND es consecuencia del teorema de DeMorgan y de la convención de que el indicador de negación denota complementación. Los dos símbolos gráficos son útiles en el análisis y diseño de circuitos NAND. Si se usan ambos símbolos en el mismo diagrama, decimos que el circuito está en notación mixta.

Implementación de dos niveles

La implementación de funciones booleanas con compuertas NAND requiere expresar la función en forma de suma de productos. Para ver la relación entre una expresión de suma de productos y su implementación NAND equivalente, consideremos los diagramas lógicos de la figura 3-20. Los tres diagramas son equivalentes e implementan la función

$$F = AB + CD$$

En a), la función se implementa con compuertas AND y OR. En (b), las compuertas AND se han sustituido por compuertas NAND y la compuerta OR se sustituyó por una compuerta NAND representada por un símbolo gráfico invertir-OR. Recuerde que una burbuja denota

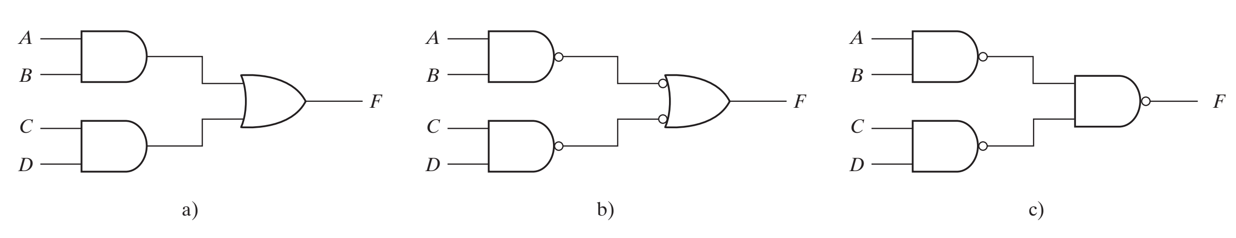


FIGURA 3-20
Tres formas de implementar $F = AB + CD$

complementación y que dos burbujas en una misma línea representan doble complementación, y pueden eliminarse. Si se quitan las burbujas de las compuertas de b) se obtiene el circuito de a). Por tanto, los dos diagramas implementan la misma función y son equivalentes.

En la figura 3-20c), se ha representado la compuerta NAND final con un símbolo gráfico AND-invertir. Al dibujar diagramas lógicos NAND, los circuitos que se muestran en b) o c) son aceptables. El de b) usa notación mixta y representa una relación más directa con la expresión booleana que implementa. La implementación NAND de la figura 3-20c) se puede verificar algebraicamente. La función que implementa se convierte fácilmente a suma de productos aplicando el teorema de DeMorgan:

$$F = ((AB)'(CD)')' = AB + CD$$

EJEMPLO 3-10

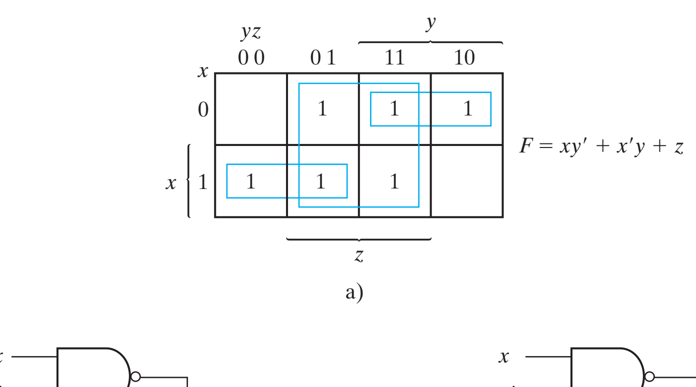
Implemente la función booleana siguiente con compuertas NAND:

$$F(x, y, z) = (1, 2, 3, 4, 5, 7)$$

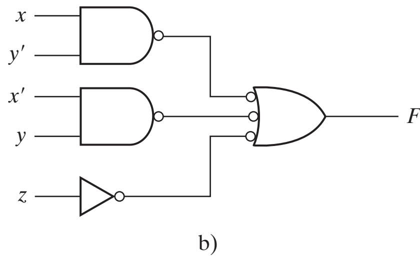
El primer paso es simplificar la función como suma de productos. Esto se hace con el mapa de la figura 3-21a), del cual se obtiene la función simplificada

$$F = xy' + x'y + z$$

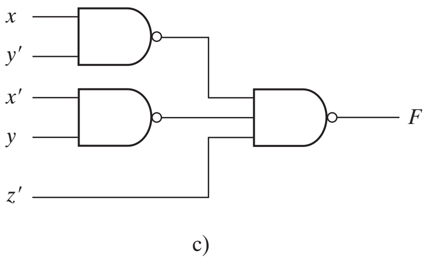
La implementación NAND de dos niveles se presenta en la figura 3-21b) en notación mixta. Observe que la entrada z necesita una compuerta NAND de una sola entrada (inversor) para compensar la burbuja de la compuerta del segundo nivel. En la figura 3-21c) se presenta otra forma de dibujar el diagrama lógico. Aquí todas las compuertas NAND se representan con el



a)



b)



c)

FIGURA 3-21
Solución del ejemplo 3-10

mismo símbolo gráfico. El inversor con entrada z se ha eliminado, pero la variable de entrada se ha complementado y se denota con z'.

■

El procedimiento descrito en el ejemplo anterior sugiere que es factible implementar una función booleana con dos niveles de compuertas NAND. El procedimiento para obtener el diagrama lógico a partir de una función booleana es el siguiente:

1. Simplificar la función y expresarla como suma de productos.
2. Incluir una compuerta NAND por cada término de producto que tenga por lo menos dos literales. Las entradas de cada compuerta NAND serán las literales del término. Esto constituye un grupo de compuertas de primer nivel.
3. Incluir una sola compuerta en el segundo nivel, empleando el símbolo gráfico AND-invertir o invertir-OR, cuyas entradas provienen de las salidas de las compuertas de primer nivel.
4. Los términos con una sola literal requerirán un inversor en el primer nivel, pero si esa literal solitaria está complementada, se le podrá conectar directamente a una entrada de la compuerta NAND del segundo nivel.

Circuitos NAND multinivel

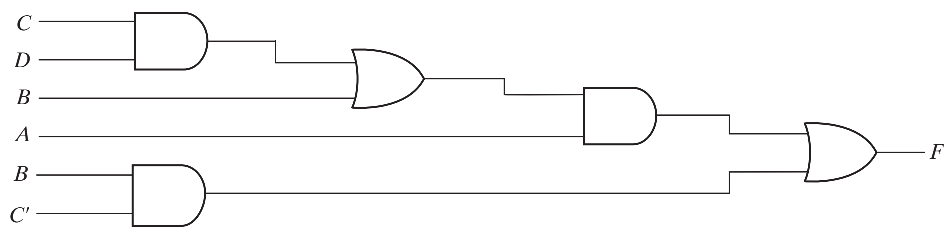
La forma estándar de expresar funciones booleanas da pie a una implementación de dos niveles. Hay ocasiones en que el diseño de sistemas digitales produce estructuras con tres o más niveles de compuertas. El procedimiento más común para diseñar circuitos multinivel es expresar la función booleana en términos de operaciones AND, OR y complemento. Entonces, la función podrá implementarse con compuertas AND y OR. Luego, si es necesario, se le puede convertir en un circuito con puras compuertas NAND. Considere, por ejemplo, la función booleana:

$$F = A(CD + B) + BC'$$

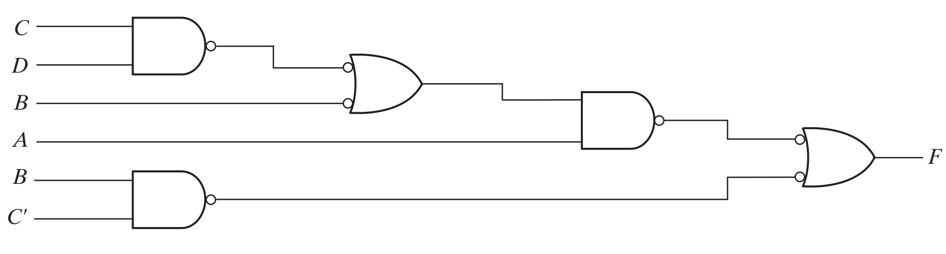
Aunque es posible quitar los paréntesis y reducir la expresión a una forma estándar de suma de productos, preferimos implementarla como circuito multinivel como ilustración. La implementación AND-OR se ilustra en la figura 3-22a). El circuito tiene cuatro niveles de compuertas. El primer nivel posee dos compuertas AND. El segundo tiene una compuerta OR seguida de una compuerta AND en el tercer nivel y una compuerta OR en el cuarto nivel. Los diagramas lógicos con un patrón de niveles alternos de compuertas AND y OR se pueden convertir fácilmente en un circuito NAND utilizando la notación mixta. Esto se muestra en la figura 3-22b). El procedimiento consiste en sustituir cada compuerta AND por un símbolo gráfico AND-invertir, y cada compuerta OR, por un símbolo gráfico invertir-OR. El circuito NAND realizará la misma lógica que el diagrama AND-OR siempre que haya dos burbujas sobre la misma línea. La burbuja asociada a la entrada B produce una complementación adicional, que debe compensarse cambiando la literal de entrada a B'.

El procedimiento general para convertir un diagrama AND-OR multinivel en un diagrama NAND con notación mixta es el siguiente:

1. Convertir todas las compuertas AND en compuertas NAND con símbolos gráficos AND-invertir.
2. Convertir todas las compuertas OR en compuertas NAND con símbolos gráficos invertir-OR.



a) Compuertas AND-OR



b) Compuertas NAND

FIGURA 3-22
Implementación de $F = A(CD + B) + BC'$

3. Revisar todas las burbujas del diagrama. Por cada burbuja que no esté compensada por otra sobre la misma línea, hay que insertar un inversor (compuerta NAND de una sola entrada) o complementar la literal de entrada.

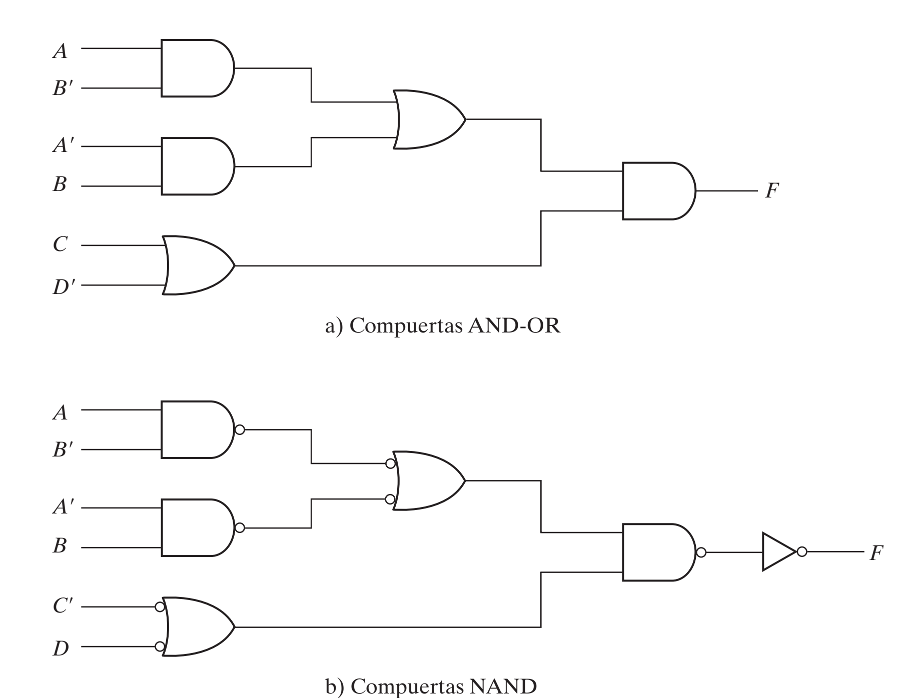
Como ejemplo adicional, consideremos la función booleana multinivel

$$F = (AB' + A'B)(C + D')$$

La implementación AND-OR se presenta en la figura 3-23a) con tres niveles de compuertas. La conversión a NAND con notación mixta aparece en la parte b) del diagrama. Las dos burbujas adicionales asociadas a las entradas C y D' hacen que estas dos literales se complementen a C' y D. La burbuja en la compuerta NAND de salida complementa el valor de salida, por lo que necesitamos insertar una compuerta inversora en la salida para complementar otra vez la señal y obtener el valor original.

Implementación NOR

La operación NOR es el dual de la operación NAND. Por tanto, todos los procedimientos y reglas para la lógica NOR son el dual de los procedimientos y reglas correspondientes que se han desarrollado para la lógica NAND. La compuerta NOR es otra compuerta universal que sirve para implementar cualquier función booleana. La implementación de las operaciones de complemento, OR y AND con compuertas NOR se aprecia en la figura 3-24. La operación de complemento se obtiene de una compuerta NOR con una sola entrada que se comporta exac-



a) Compuertas AND-OR

b) Compuertas NAND

FIGURA 3-23
Implementación de $F = (AB' + A'B)(C + D')$

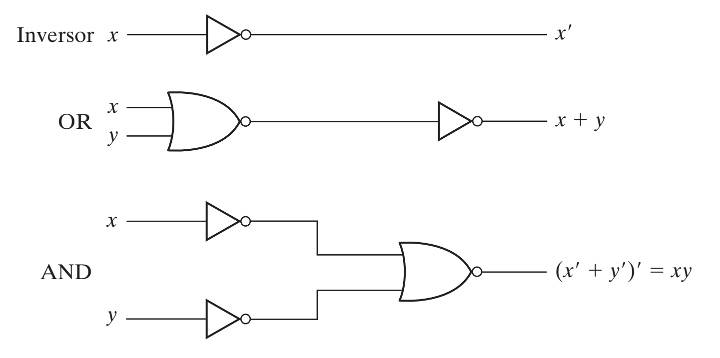


FIGURA 3-24
Operaciones lógicas con compuertas NOR

tamente como un inversor. La operación OR requiere dos compuertas NOR y la AND, una compuerta NOR con inversores en cada entrada.

Los dos símbolos gráficos de la notación mixta se muestran en la figura 3-25. El símbolo OR-invertir define la operación NOR como un OR seguido de un complemento. El símbolo invertir-AND complementa cada una de las entradas y luego realiza una operación AND. Los dos símbolos designan la misma operación NOR y son lógicamente idénticos por el teorema de DeMorgan.

$(x + y + z)'$ $\quad$ $x'y'z' = (x + y + z)'$

a) OR-invertir $\quad$ b) invertir-AND

FIGURA 3-25
Dos símbolos gráficos para la compuerta NOR

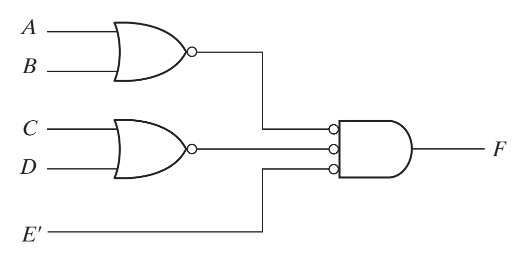


FIGURA 3-26
Implementación de $F = (A + B)(C + D)E$

Una implementación de dos niveles con compuertas NOR requiere simplificar la función en forma de producto de sumas. Recuerde que la expresión simplificada de producto de sumas se obtiene del mapa combinando los ceros y complementando. Una expresión en producto de sumas se implementa con un primer nivel de compuertas OR que produce los términos de suma, seguido de una compuerta AND de segundo nivel que da el producto. La transformación del diagrama OR-AND en un diagrama NOR se logra cambiando las compuertas OR por compuertas NOR representadas por símbolos gráficos OR-invertir, y la compuerta AND, por una compuerta NOR representada por un símbolo gráfico invertir-AND. Si la compuerta de segundo nivel tiene como entrada un término de una sola literal, ésta deberá complementarse. La figura 3-26 muestra la implementación NOR de una función expresada como producto de sumas:

$$F = (A + B)(C + D)E$$

El patrón OR-AND se detecta fácilmente quitando los pares de burbujas que estén sobre la misma línea. La variable E se complementa para compensar la tercera burbuja en la entrada de la compuerta de segundo nivel.

El procedimiento para convertir un diagrama AND-OR multinivel en un diagrama sólo NOR es similar al que se utilizó para las compuertas NAND. En el caso de NOR, hay que sustituir cada compuerta OR por un símbolo OR-invertir, y cada compuerta AND, por un símbolo invertir-AND. Toda burbuja que no esté compensada por otra burbuja en la misma línea necesitará un inversor, o se deberá complementar la literal de entrada.

La transformación del diagrama AND-OR de la figura 3-23a) en un diagrama NOR se ilustra en la figura 3-27. La función booleana para este circuito es

$$F = (AB' + A'B)(C + D')$$

El diagrama AND-OR equivalente se deduce del diagrama NOR quitando todas las burbujas. Para compensar las burbujas en cuatro entradas, es preciso complementar las literales de entrada correspondientes.

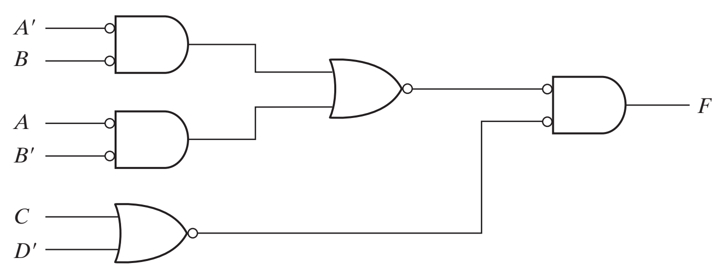


FIGURA 3-27
Implementación de $F = (AB' + A'B)(C + D')$ con compuertas NOR

3-7 OTRAS IMPLEMENTACIONES DE DOS NIVELES

Los tipos de compuertas que más comúnmente se encuentran en los circuitos integrados son las NAND y NOR. Por ello, las implementaciones de lógica NAND y NOR son las más importantes en la práctica. Algunas compuertas NAND o NOR (pero no todas) contemplan la posibilidad de una conexión con alambre entre las salidas de dos compuertas para formar una función lógica específica. Este tipo de lógica se llama *lógica alambrada* (*wired*, en inglés). Por ejemplo, si se conectan entre sí compuertas NAND TTL de colector abierto, efectúan la lógica AND alambrada. (La compuerta TTL de colector abierto se representa en la figura 10-11 del capítulo 10.) La lógica AND alambrada efectuada con dos compuertas NAND se muestra en la figura 3-28a). La compuerta AND se dibuja con líneas que pasan por el centro de la compuerta, para distinguirla de una compuerta convencional. La compuerta AND alambrada no es una compuerta física, sino un símbolo que designa la función obtenida de la conexión alambrada que se indica. La función lógica implementada por el circuito de la figura 3-28a) es

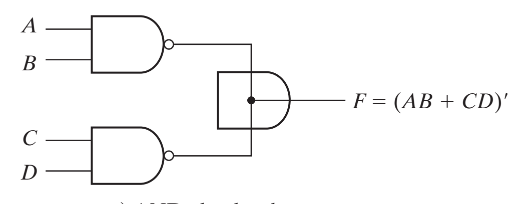
$$F = (AB)' \cdot (CD)' = (AB + CD)'$$

y se denomina función AND-OR-INVERT.

Asimismo, la salida NOR de compuertas ECL se puede vincular para desempeñar una función OR alambrada. La función lógica implementada por el circuito de la figura 3-28b) es

$$F = (A + B)' + (C + D)' = \left[(A + B)(C + D)\right]'$$

y se denomina función OR-AND-INVERT.



a) AND alambrado en compuertas NAND TTL de colector abierto

(AND-OR-INVERT)

A
B
C
D
F = [(A + B) (C + D)]′

b) OR alambrado en compuertas ECL

(OR-AND-INVERT)

FIGURA 3-28
Lógica alambrada

Una compuerta de lógica alambrada no produce una compuerta física de segundo nivel porque no es más que una conexión de alambres. No obstante, en nuestras explicaciones, consideraremos los circuitos de la figura 3-28 como implementaciones de dos niveles. El primer nivel consiste en compuertas NAND (o NOR) y el segundo tiene una sola compuerta AND (u OR). La conexión alambrada dentro del símbolo gráfico se omitirá en explicaciones posteriores.

Formas no degeneradas

Desde un punto de vista teórico, resulta interesante averiguar cuántas combinaciones de compuertas de dos niveles puede haber. Consideraremos cuatro tipos de compuertas: AND, OR, NAND y NOR. Si asignamos un tipo de compuerta al primer nivel y un tipo al segundo nivel, encontramos que hay 16 posibles combinaciones de formas de dos niveles. (Se puede usar el mismo tipo en el primer nivel y en el segundo, como en la implementación NAND-NAND.) Ocho de esas combinaciones se consideran formas *degeneradas* porque degeneran a una sola operación. Esto se percibe en un circuito con compuertas AND en el primer nivel y una compuerta AND en el segundo nivel. La salida del circuito no es más que la función AND de todas las variables de entrada. Las otras ocho formas *no degeneradas* producen una implementación de suma de productos o producto de sumas. Las ocho formas no degeneradas son:

AND-OR	OR-AND
NAND-NAND	NOR-NOR
NOR-OR	NAND-AND
OR-NAND	AND-NOR

La primera compuerta indicada en cada una de las formas constituye el primer nivel de la implementación. La segunda compuerta es una sola, colocada en el segundo nivel. Hemos presentado las formas en pares, de modo que las dos formas de cada línea son una el dual de la otra.

Las formas AND-OR y OR-AND son las formas básicas de dos niveles que vimos en la sección 3-4. Presentamos las formas NAND-NAND y NOR-NOR en la sección 3-6. En esta sección investigaremos las otras cuatro formas.

Implementación AND-OR-INVERT

Las dos formas NAND-AND y AND-NOR son equivalentes y podemos verlas juntas. Ambas efectúan la función AND-OR-INVERT, como se muestra en la figura 3-29. La forma AND-NOR se parece a la forma AND-OR pero con una inversión indicada por la burbuja en la salida de la compuerta NOR. Esta forma implementa la función

$$F = (AB + CD + E)'$$

Al utilizar el símbolo gráfico alterno para la compuerta NOR, se obtiene el diagrama de la figura 3-29b). Advierta que la variable sola *E no* se complementa porque el único cambio que se efectúa es en el símbolo gráfico de la compuerta NOR. Ahora pasamos la burbuja de la terminal de entrada de la compuerta de segundo nivel a las terminales de salida de las compuertas de primer nivel. Se necesita un inversor para la variable sola, como compensación de la

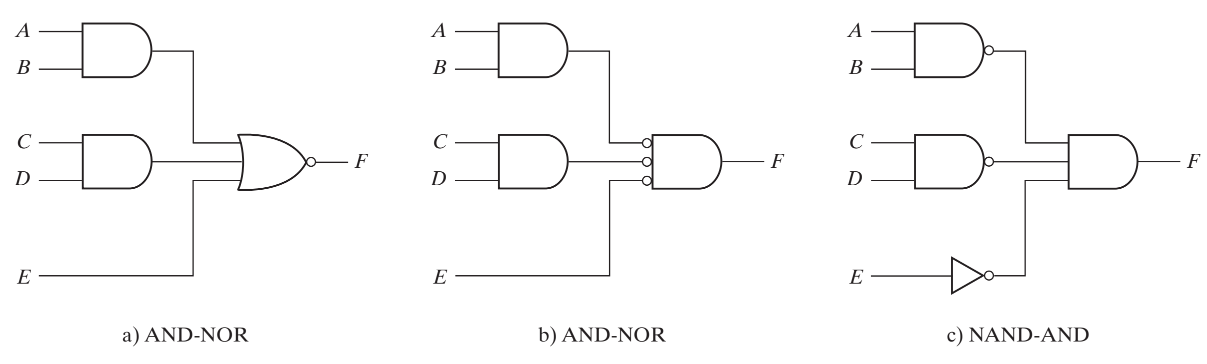


FIGURA 3-29
Circuitos AND-OR-INVERT; $F = (AB + CD + E)'$

burbuja. O bien, podemos quitar el inversor si la entrada E se complementa. El circuito de la figura 3-29c) es una forma NAND-AND, y en la figura 3-28 vimos que implementa la función AND-OR-INVERT.

Una implementación AND-OR requiere una expresión en forma de suma de productos. La implementación AND-OR-INVERT es similar, excepto por la inversión. Por tanto, si simplificamos el *complemento* de la función en forma de suma de productos (combinando los ceros del mapa), podremos implementar F' con la parte AND-OR de la función. Cuando F' pase por la obligada inversión de salida (la parte INVERT), generará la salida de la función F. Más adelante se presentará un ejemplo de la implementación AND-OR-INVERT.

Implementación OR-AND-INVERT

Las formas OR-NAND y NOR-OR efectúan la función OR-AND-INVERT. Esto se observa en la figura 3-30. La forma OR-NAND se parece a la OR-AND, excepto por la inversión efectuada por la burbuja de la compuerta NAND. Esta forma implementa la función

$$F = \left[(A + B)(C + D)E\right]'$$

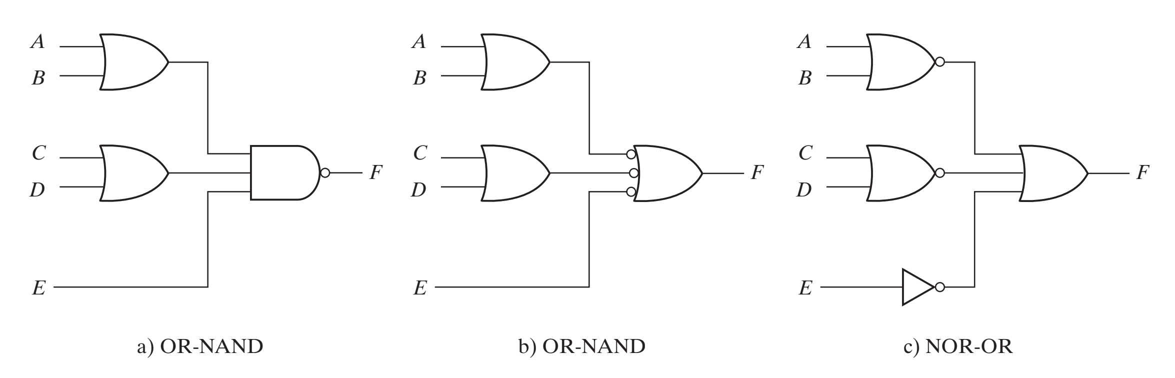


FIGURA 3-30
Circuitos OR-AND-INVERT; $F = [(A + B)(C + D)E]'$

Tabla 3-3
Implementación con otras formas de dos niveles

Forma no degenerada equivalente		Implementa la función	Simplificar F' en	Para obtener como salida
a)	**b)***			
AND-NOR	NAND-AND	AND-OR-INVERT	Suma de productos combinando los ceros del mapa	F
OR-NAND	NOR-OR	OR-AND-INVERT	Producto de sumas combinando los unos del mapa y complementando después	F

* La forma b) requiere un inversor en los términos de una sola literal.

Si se utiliza el símbolo gráfico alterno para la compuerta NAND, se obtiene el diagrama de la figura 3-30b). El circuito de c) se obtiene pasando las burbujas de las entradas de la compuerta de segundo nivel a las salidas de las compuertas de primer nivel. El circuito de la figura 3-30c) es una forma NOR-OR y en la figura 3-28 se demostró que implementa la función OR-AND-INVERT.

La implementación OR-AND-INVERT requiere una expresión en forma de producto de sumas. Si el complemento de la función se simplifica en producto de sumas, se podrá implementar F' con la parte OR-AND de la función. Una vez que F' pase por la parte INVERT, tendremos el complemento de F', o F, en la salida.

Resumen tabular y ejemplo

En la tabla 3-3 se resumen los procedimientos para implementar una función booleana en cualquiera de las cuatro formas de dos niveles. Debido a la parte INVERT en todos los casos, conviene usar la simplificación de la función, F' (el complemento). Al implementar F' en una de estas formas, se obtiene el complemento de la función en la forma AND-OR u OR-AND. Las cuatro formas de dos niveles invierten esta función para dar como salida el complemento de F', que es la salida normal F.

EJEMPLO 3-11

Implemente la función de la figura 3-31a) con las cuatro formas de dos niveles presentadas en la tabla 3-3.

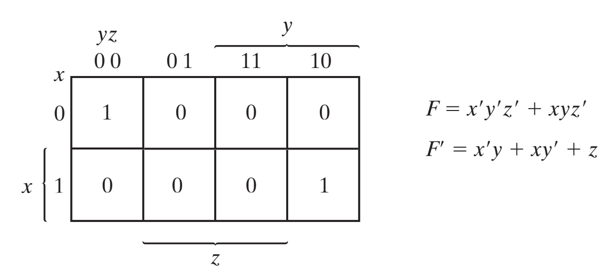
El complemento de la función se simplifica en forma de suma de productos combinando los ceros del mapa:

$$F' = x'y + xy' + z$$

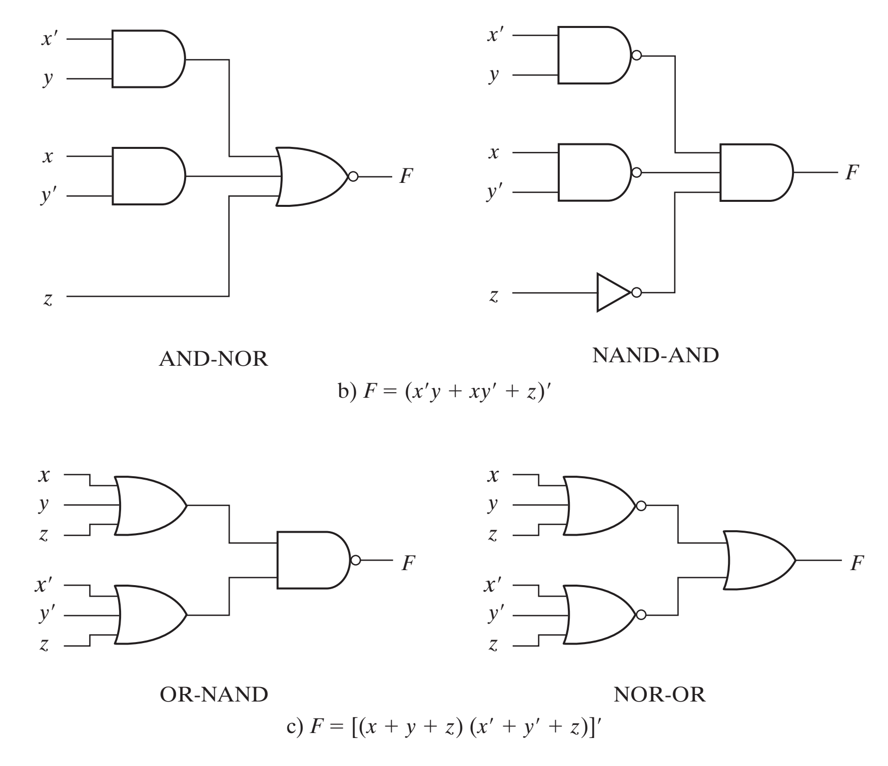
La salida normal de esta función se expresa así:

$$F = (x'y + xy' + z)'$$

que está en la forma AND-OR-INVERT. En la figura 3-31b) se muestran las implementaciones AND-NOR y NAND-AND. Advierta que se necesita una compuerta NAND de una sola en-



a) Simplificación por mapa en forma de suma de productos



b) $F = (x'y + xy' + z)'$

c) $F = [(x + y + z)(x' + y' + z)]'$

FIGURA 3-31
Otras implementaciones de dos niveles

trada (un inversor) en la implementación NAND-AND, pero no en el caso AND-NOR. Es posible quitar ese inversor si se aplica la variable de entrada z' en lugar de z.

Las formas OR-AND-INVERT requieren una expresión simplificada del complemento de la función en forma de producto de sumas. Para obtener esta expresión, primero hay que combinar los unos del mapa:

$$F = x'y'z' + xyz'$$

Luego se obtiene el complemento de la función

$$F' = (x + y + z)(x' + y' + z)$$

Ahora se expresa la salida normal de la función, F, en la forma

$$F = [(x + y + z)(x' + y' + z)]'$$

que está en la forma OR-AND-INVERT. A partir de esta expresión, se implementa la función en las formas OR-NAND y NOR-OR, como se indica en la figura 3-31c).

■

3-8 FUNCIÓN OR EXCLUSIVO

La función OR exclusivo (XOR), denotada por el símbolo $\oplus$, es una operación lógica que efectúa la operación booleana siguiente:

$$x \oplus y = xy' + x'y$$

Es igual a 1 si sólo x es igual a 1 o sólo y es igual a 1, pero no si ambas son 1. El NOR exclusivo, también llamado equivalencia, realiza la operación booleana siguiente:

$$(x \oplus y)' = xy + x'y'$$

Es igual a 1 si tanto x como y son 1 o si ambas son 0. Se puede demostrar que el NOR exclusivo es el complemento del OR exclusivo con la ayuda de una tabla de verdad o por manipulación algebraica:

$$(x \oplus y)' = (xy' + x'y)' = (x' + y)(x + y') = xy + x'y'$$

Se cumplen las identidades siguientes para la operación de OR exclusivo:

$$x \oplus 0 = x$$
$$x \oplus 1 = x'$$
$$x \oplus x = 0$$
$$x \oplus x' = 1$$
$$x \oplus y' = x' \oplus y = (x \oplus y)'$$

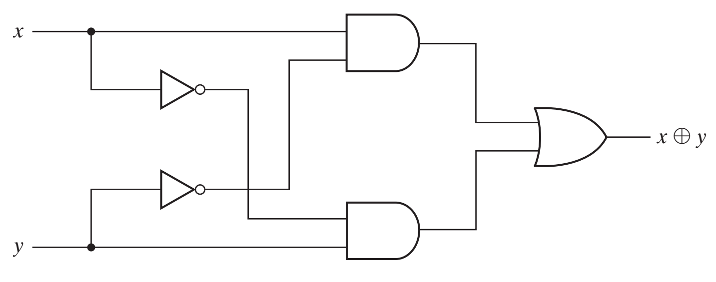
Es factible demostrar cualquiera de estas identidades con una tabla de verdad o sustituyendo la operación $\oplus$ por su expresión booleana equivalente. También puede demostrarse que la operación OR exclusivo es tanto conmutativa como asociativa; es decir,

$$A \oplus B = B \oplus A$$

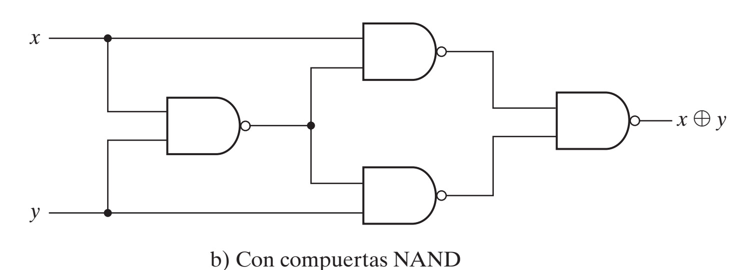
y

$$(A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C$$

Esto significa que las dos entradas de una compuerta OR exclusivo son intercambiables sin afectar a la operación. También implica que podemos evaluar una operación OR exclusivo de tres variables en cualquier orden, así que las tres o más variables se expresan sin paréntesis. Esto implicaría la posibilidad de usar compuertas OR exclusivo con tres o más entradas. Sin embargo, es difícil fabricar con hardware compuertas OR exclusivo de múltiples entradas. De hecho, incluso las funciones de dos entradas suelen construirse con otros tipos de compuertas.



a) Con compuertas AND-OR-NOT



b) Con compuertas NAND

FIGURA 3-32
Implementaciones del OR exclusivo

Construimos una función OR exclusivo de dos entradas a partir de compuertas convencionales usando dos inversores, dos compuertas AND y una compuerta OR, como se indica en la figura 3-32a). La figura 3-32b) muestra la implementación del OR exclusivo con cuatro compuertas NAND. La primera compuerta NAND efectúa la operación $(xy)' = (x' + y')$. El otro circuito NAND de dos niveles produce la suma de productos de sus entradas:

$$(x' + y')x + (x' + y')y = xy' + x'y = x \oplus y$$

Son pocas las funciones booleanas que se pueden expresar en términos de operaciones OR exclusivo. No obstante, esta función surge con mucha frecuencia durante el diseño de sistemas digitales. Tiene especial utilidad en operaciones aritméticas y en circuitos para detectar y corregir errores.

Función impar

La operación OR exclusivo con tres o más variables se convierte en una función booleana ordinaria sustituyendo el símbolo $\oplus$ por su expresión booleana equivalente. En particular, el caso de tres variables se puede convertir en una expresión booleana así:

$$\begin{aligned} A \oplus B \oplus C &= (AB' + A'B)C' + (AB + A'B')C \\ &= AB'C' + A'BC' + ABC + A'B'C \\ &= \Sigma(1, 2, 4, 7) \end{aligned}$$

La expresión booleana indica claramente que la función OR exclusivo de tres variables es igual a 1 si sólo una variable es 1 o si las tres variables son 1. A diferencia del caso de dos variables,

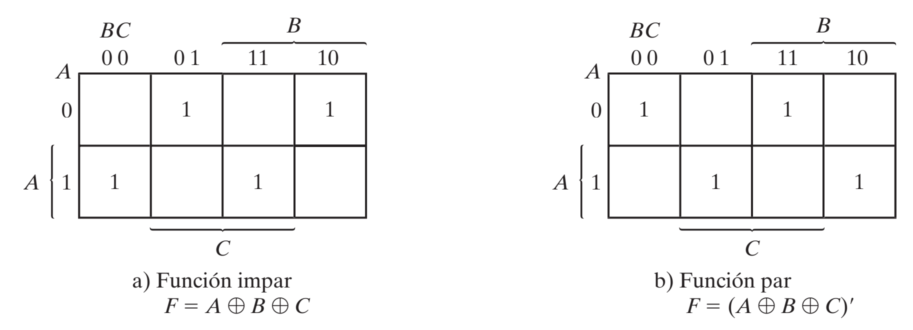


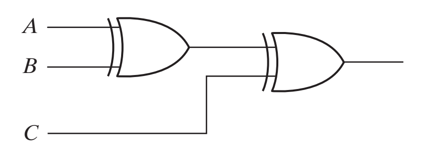
FIGURA 3-33
Mapa para una función OR exclusivo de tres variables

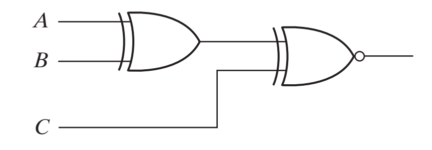
en el que sólo una variable debe ser igual a 1, en el caso de tres o más variables el requisito es que un número impar de variables sea igual a 1. Por consiguiente, la operación OR exclusivo de múltiples variables se define como *función impar*.

La función booleana obtenida de la operación OR exclusivo de tres variables se expresa como la suma lógica de cuatro minitérminos cuyos valores numéricos binarios son 001, 010, 100 y 111. Todos estos números binarios tienen un número impar de unos. Los otros cuatro minitérminos no incluidos en la función son 000, 011, 101 y 110, y tienen un número par de unos en su valor numérico binario. En general, una función OR exclusivo de n variables es una función impar definida como la suma lógica de los $2^n/2$ minitérminos cuyos valores numéricos binarios tienen un número impar de unos.

La definición de función impar queda más clara si se grafica en un mapa. La figura 3-33a) muestra el mapa para la función OR exclusivo de tres variables. Los cuatro minitérminos de la función están separados por una distancia unitaria. La función impar se identifica a partir de los cuatro minitérminos cuyos valores binarios tienen un número impar de unos. El complemento de una función impar es una función par. Como se aprecia en la figura 3-33b), la función par de tres variables es 1 cuando un número par de variables es igual a 1 (incluida la condición en la que ninguna de las variables es igual a 1).

La función impar de tres entradas se implementa con compuertas OR exclusivo de dos entradas, como se observa en la figura 3-34a). El complemento de una función impar se obtiene sustituyendo la compuerta de salida por una compuerta NOR exclusivo, como se indica en la figura 3-34b).

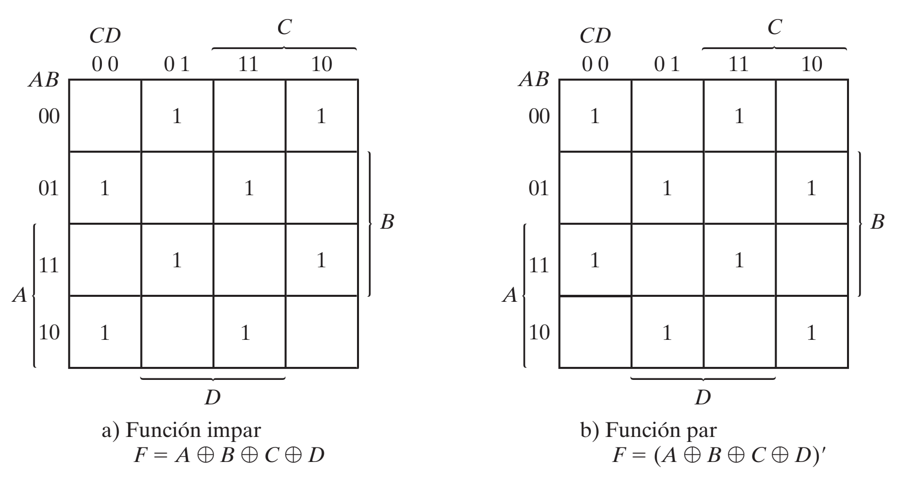




a) Función impar de tres entradas

b) Función par de tres entradas

FIGURA 3-34
Diagrama lógico de funciones impar y par



a) Función impar
$F = A \oplus B \oplus C \oplus D$

b) Función par
$F = (A \oplus B \oplus C \oplus D)'$

FIGURA 3-35
Mapa de una función OR exclusivo de cuatro variables

Considere ahora la operación OR exclusivo de cuatro variables. Por manipulación algebraica, se obtiene la suma de minitérminos para esta función:

$$
\begin{aligned}
A \oplus B \oplus C \oplus D &= (AB' + A'B) \oplus (CD' + C'D) \\
&= (AB' + A'B)(CD + C'D') + (AB + A'B')(CD' + C'D) \\
&= \sum(1, 2, 4, 7, 8, 11, 13, 14)
\end{aligned}
$$

Una función booleana de cuatro variables tiene 16 minitérminos. La mitad de ellos tiene valores numéricos binarios con un número impar de unos; la otra mitad tiene valores numéricos binarios con un número par de unos. Al graficar la función en el mapa, el valor numérico binario de un minitérmino se deduce de los números de fila y columna del cuadrado que representa al minitérmino. El mapa de la figura 3-35a) corresponde a la función OR exclusivo de cuatro variables. Es una función impar porque los valores binarios de todos los minitérminos tienen un número impar de unos. El complemento de una función impar es una función par. Como se observa en la figura 3-35b), la función par de cuatro variables es igual a 1 cuando un número par de variables es igual a 1.

Generación y verificación de paridad

Las funciones OR exclusivo son muy útiles en los sistemas que requieren códigos para detectar y corregir errores. Como se explicó en la sección 1-7, se utiliza un bit de paridad para detectar errores durante la transmisión de información binaria. Un bit de paridad es un bit adicional que se incluye con el mensaje binario de modo que el número total de unos sea impar o par. El mensaje, con el bit de paridad incluido, se transmite y luego se verifica en el extremo receptor para comprobar que no haya habido errores. Se detecta un error si la paridad recibida no corresponde con la transmitida. El circuito que genera el bit de paridad en el transmisor se denomina *generador de paridad*. El que comprueba la paridad en el receptor se llama *verificador de paridad*.

Tabla 3-4
Tabla de verdad de un generador de paridad par

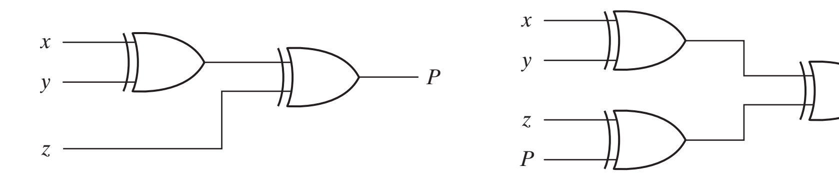
Mensaje de tres bits			Bit de paridad
x	***y***	***z***	***P***
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Como ejemplo, considere un mensaje de tres bits que se transmitirá junto con un bit de paridad par. La tabla 3-4 representa la tabla de verdad para el generador de paridad. Los tres bits —x, y y z— constituyen el mensaje y son las entradas del circuito. La salida es el bit de paridad, P. Si se usa paridad par, el bit P generado deberá hacer que el número total de unos (incluido P) sea par. Por la tabla de verdad, vemos que P constituye una función impar porque es igual a 1 para todos los minitérminos cuyo valor numérico binario tiene un número impar de unos. Por tanto, P se expresa como una función OR exclusivo de tres variables:

$$P = x \oplus y \oplus z$$

El diagrama lógico del generador de paridad se muestra en la figura 3-36a).

Los tres bits del mensaje, junto con el bit de paridad, se transmiten a su destino, donde se aplican a un circuito verificador de paridad para detectar posibles errores en la transmisión. Puesto que la información se transmitió con paridad par, los cuatro bits recibidos deberán tener un número par de unos. Si los cuatro bits recibidos tienen un número impar de unos, querrá decir que hubo un error de transmisión, pues por lo menos un bit habrá cambiado de valor durante la transmisión. La salida del verificador de paridad, denotada por C, será igual a 1 si hubo un error, es decir, si los cuatro bits recibidos tienen un número impar de unos. La tabla 3-5 es la tabla de verdad del verificador de paridad par. En ella se advierte que la función C consiste en los ocho minitérminos cuyo valor numérico binario posee un número impar de unos. Esto



a) Generador de paridad par de tres bits

b) Verificador de paridad par de cuatro bits

FIGURA 3-36
Diagrama lógico de un generador y un verificador de paridad

Tabla 3-5
Tabla de verdad de un verificador de paridad par

Cuatro bits recibidos				Verificador de errores de paridad
x	*y*	*z*	*P*	*C*
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

corresponde al mapa de la figura 3-35a), que representa una función impar. El verificador de paridad se implementa con compuertas OR exclusivo:

$$C = x \oplus y \oplus z \oplus P$$

El diagrama lógico del verificador de paridad se muestra en la figura 3-36b).

Vale la pena señalar que el generador de paridad se implementa con el circuito de la figura 3-36b) si la entrada P se conecta a 0 lógico y la salida se designa como P. Ello se debe a que $z \oplus 0 = z$, con lo que el valor de z pasa inalterado por la compuerta. La ventaja de esto es que se puede usar el mismo circuito tanto para generar como para verificar la paridad.

Por el ejemplo anterior, es obvio que los circuitos para generar y verificar paridad siempre tienen una función de salida que incluye la mitad de los minitérminos, aquellos cuyo valor numérico tiene un número impar (o par) de unos. Por consiguiente, se les puede implementar con compuertas OR exclusivo. Una función con un número par de unos es el complemento de una función impar y se implementa con compuertas OR exclusivo, salvo que la compuerta asociada con la salida debe ser un NOR exclusivo para realizar la complementación requerida.

3-9 LENGUAJE DE DESCRIPCIÓN DE HARDWARE (HDL)

Los lenguajes de descripción de hardware son lenguajes que describen el hardware de los sistemas digitales en forma textual. Se parecen a los lenguajes de programación, pero están orientados específicamente a la descripción de las estructuras y el comportamiento del hardware.

Sirven para representar diagramas lógicos, expresiones booleanas y otros circuitos digitales más complejos. Como lenguaje de *documentación*, un HDL sirve para representar y documentar sistemas digitales en una forma susceptible de ser leída tanto por personas como por computadoras, y es apropiado como lenguaje de intercambio entre diseñadores. El contenido en HDL se puede almacenar, recuperar y procesar fácil y eficazmente con software de computadora. Hay dos aplicaciones del procesamiento de HDL: simulación y síntesis.

La *simulación lógica* es la representación de la estructura y el comportamiento de un sistema lógico digital empleando una computadora. El simulador interpreta la descripción en HDL y produce una salida comprensible, digamos un diagrama de temporización, que predice la forma en que se comportará el hardware antes de que se fabrique físicamente. La simulación permite detectar errores funcionales de un diseño sin tener que crear el circuito físico. Los errores detectados durante la simulación se corrigen modificando los enunciados HDL apropiados. El estímulo que prueba la funcionalidad del diseño se denomina *conjunto de pruebas*. Así, para simular un sistema digital, el diseño se describe primero en HDL y luego se verifica simulando el diseño y verificándolo con un conjunto de pruebas, que también está escrito en HDL.

La *síntesis lógica* es el proceso de deducir una lista de componentes y sus interconexiones (lo que se conoce como *lista del circuito*) a partir del modelo de un sistema digital descrito en HDL. La lista del circuito en el nivel de compuertas sirve para fabricar un circuito integrado o para diagramar una tarjeta de circuitos impresos. La síntesis lógica es similar a la compilación de un programa en un lenguaje de alto nivel convencional. La diferencia es que, en lugar de producir código objeto, la síntesis lógica produce una base de datos con instrucciones para fabricar un circuito digital físico que implementa los enunciados descritos por el código HDL. La síntesis lógica se basa en procedimientos formales exactos que implementan circuitos digitales, y es la parte de un diseño digital que se automatiza con software.

Hay muchos HDL exclusivos en la industria, desarrollados por empresas que diseñan o ayudan a diseñar circuitos integrados. El IEEE (Instituto de Ingenieros en Electricidad y Electrónica) apoya dos HDL estándar: VHDL y Verilog HDL. VHDL es un lenguaje cuyo uso exige el Departamento de Defensa de EU y que inicialmente usaban los contratistas de esa dependencia, aunque ahora se le usa comercialmente y en universidades de investigación. Verilog nació como un HDL exclusivo, promovido por una compañía llamada Cadence Data Systems, pero Cadence transfirió el control de Verilog a un consorcio de empresas y universidades llamado Open Verilog International (OVI). VHDL es un lenguaje más difícil de aprender y usar que Verilog, y es por ello que se emplea Verilog en este libro. No obstante, las descripciones en Verilog HDL que aquí se presentan no pretenden enseñar Verilog, sino más bien introducir el concepto de representación de sistemas digitales asistida por computadora utilizando un lenguaje típico de descripción de hardware.

Representación de módulos

La sintaxis de Verilog describe con precisión las construcciones válidas que se pueden usar en el lenguaje. En particular, Verilog utiliza cerca de 100 palabras clave: identificadores predefinidos, en minúsculas, que definen las construcciones del lenguaje. Como ejemplos de palabras clave podemos citar module, endmodule, input, output, wire, and, or, not, etcétera. Todo texto comprendido entre dos diagonales (//) y el fin de la línea se interpreta como un comentario. Se hace caso omiso de los espacios en blanco y se hace distinción entre mayúsculas y minúsculas. El bloque de construcción en Verilog es el módulo. Declaramos un módulo con la

palabra clave **module** y marcamos su final con la palabra clave **endmodule.** A continuación se presentaa un ejemplo sencillo para ilustrar algunos aspectos del lenguaje.

En el ejemplo HDL 3-1 se hace una descripción en HDL del circuito de la figura 3-37. La línea que inicia con dos diagonales es un comentario que explica la función del circuito. La segunda línea declara el módulo, e incluye un nombre y una lista de puertos. El nombre (`circuito_smpl` en este caso) es un identificador que sirve para referirse al módulo. Los identificadores son nombres que se dan a las variables para poder referirse a ellas en el diseño. Constan de caracteres alfanuméricos y el carácter de subraya (_), y hay distinción entre mayúsculas y minúsculas. Los identificadores deben iniciar con un carácter alfabético o subraya; no pueden iniciar con un número. La lista de puertos es la interfaz a través de la cual el módulo se comunica con su entorno. En este ejemplo, los puertos son las entradas y salidas del circuito. La lista de puertos se encierra en paréntesis, y se usan comas para separar los elementos de la lista. El enunciado termina con un signo de punto y coma (;). Todas las palabras clave (que deben estar en minúscula) se han imprimido aquí en negritas por claridad, pero el lenguaje no requiere esto. A continuación, las declaraciones **input** y **output** definen cuáles puertos son entradas y cuáles son salidas. Las conexiones internas se declaran como alambres. El circuito tiene una conexión interna en la terminal *e*, la cual se declara con la palabra clave **wire.** La estructura del circuito se especifica empleando las compuertas primitivas predefinidas como palabras clave. Cada declaración de compuerta consiste en un nombre opcional (como g1, g2, etcétera) seguido de la salida y las entradas de la compuerta, separadas por comas y encerradas en paréntesis. La salida siempre va primero, seguida de las entradas. Por ejemplo, la compuerta OR se llama g3, su salida es *x* y tiene como entradas *e* y *y*. La descripción del módulo termina con la palabra clave **endmodule.** Observe que cada enunciado termina con un signo de punto y coma, pero no hay un punto y coma después de **endmodule.**

Ejemplo HDL 3-1

```
//Descripción del circuito simple de la fig. 3-37
module circuito_smpl(A,B,C,x,y);
   input A,B,C;
   outputx,y;
   wire e;
   and g1(e,A,B);
   not g2(y, C);
   or g3(x,e,y);
endmodule
```

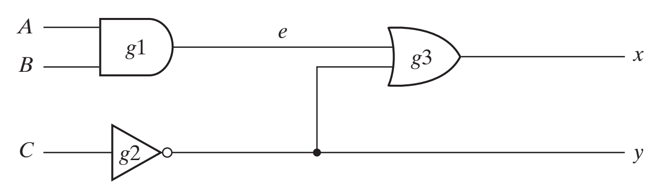


FIGURA 3-37
Circuito para ilustrar HDL

Retardos de compuerta

Cuando se usa HDL para hacer simulaciones, tal vez sea necesario especificar la magnitud del retardo que hay entre la entrada y la salida de las compuertas. En Verilog, el retardo se especifica en términos de *unidades de tiempo* y el símbolo #. La asociación de la unidad de tiempo con el tiempo físico se efectúa con la directriz de compilador **`timescale.** (Las directrices al compilador inician con el símbolo ` (acento grave).) Tales directrices se especifican antes de declarar módulos. Un ejemplo de directriz de escala de tiempo es:

`timescale 1ns/100ps

El primer número especifica la unidad de medición de los retardos. El segundo especifica la precisión del redondeo de los retardos, en este caso a 0.1 ns. Si no se especifica una escala de tiempo, el simulador utilizará por omisión cierta unidad de tiempo, por lo regular 1 ns. (1 ns $= 10^{-9}$ s). En este libro supondremos la unidad de tiempo por omisión.

El ejemplo HDL 3-2 repite la descripción del circuito simple especificando retardos para cada compuerta. Las compuertas AND, OR y NOT tienen un retardo de 30, 20 y 10 ns, respectivamente. Si se simula el circuito y las entradas cambian de 000 a 111, las salidas cambiarán como se indica en la tabla 3-6. La salida del inversor en *y* cambia de 1 a 0 después de un retardo de 10 ns. La salida de la compuerta AND en *e* cambia de 0 a 1 después de un retardo de 30 ns.

Ejemplo HDL 3-2

```
//Descripción de circuito con retardo
module circuito_con_retardo(A,B,C,x,y);
   input A,B,C;
   output x,y;
   wire e;
   and #(30) g1(e,A,B);
   or #(20) g3(x,e,y);
   not #(10) g2(y,C);
endmodule
```

Tabla 3-6
Salida de las compuertas después del retardo

	Unidades de tiempo (ns)	Entrada ABC	Salida *y e x*
Inicial	–	000	1 0 1
Cambio	–	111	1 0 1
	10	111	0 0 1
	20	111	0 0 1
	30	111	0 1 0
	40	111	0 1 0
	50	111	0 1 1

Ejemplo HDL 3-3

```
//Estímulo para el circuito simple
module stimcrct;
reg A,B,C;
wire x,y;
circuito_con_retardo ccr(A,B,C,x,y);
initial
   begin
        A = 1'b0; B = 1'b0; C = 1'b0;
     #100
        A = 1'b1; B = 1'b1; C = 1'b1;
     #100  $finish;
   end
endmodule

//Descripción de circuito con retardo
module circuito_con_retardo (A,B,C,x,y);
   input A,B,C;
   output x,y;
   wire e;
   and #(30) g1(e,A,B);
   or  #(20) g3(x,e,y);
   not #(10) g2(y,C);
endmodule
```

La salida de la compuerta OR en x cambia de 1 a 0 en $t = 30$ ns, y luego vuelve a 1 en $t = 50$ ns. En ambos casos, el cambio en la salida de la compuerta OR es resultado de un cambio en sus entradas 20 ns antes. Es evidente, por este resultado, que si bien la salida x finalmente se estabiliza en 1 después de los cambios en sus entradas, los retardos de compuerta producen un pico negativo de 20 ns antes de que eso suceda.

Al simular un circuito con HDL, es necesario aplicar entradas al circuito para que el simulador genere una respuesta de salida. Un *conjunto de pruebas* es una descripción HDL que suministra el estímulo a un diseño. Al final de la sección 4-11 se explica cómo escribir conjuntos de pruebas. Aquí se explica el procedimiento con un ejemplo sencillo sin entrar en muchos pormenores. El ejemplo HDL 3-3 muestra un conjunto de pruebas para estimular el circuito con retardo. Se incluyen dos módulos: un módulo de estímulo y el módulo que describe el circuito. El módulo de estímulo `stimcrct` no tiene puertos. Las entradas del circuito se declaran con la palabra clave **reg,** y las salidas, con la palabra clave **wire.** Se usa un ejemplar de `circuito_con_retardo` que lleva el nombre `ccr`. (La interacción entre el módulo de estímulo y el módulo de diseño se representa en la figura 4-33.) El enunciado **initial** especifica las entradas entre las palabras clave **begin** y **end.** Inicialmente, $ABC = 000$. (A, B y C se ajustan a 1′b0, lo que significa un dígito binario con valor de 0.) Después de 100 ns, las entradas cambian a $ABC = 111$. Después de otros 100 ns, termina la estimulación (**$finish** es una tarea del sistema). El diagrama de temporización resultado de la simulación aparece en la figura 3-38. La simulación tarda 200 ns en total. Las entradas A, B y C cambian de 0 a 1 des-

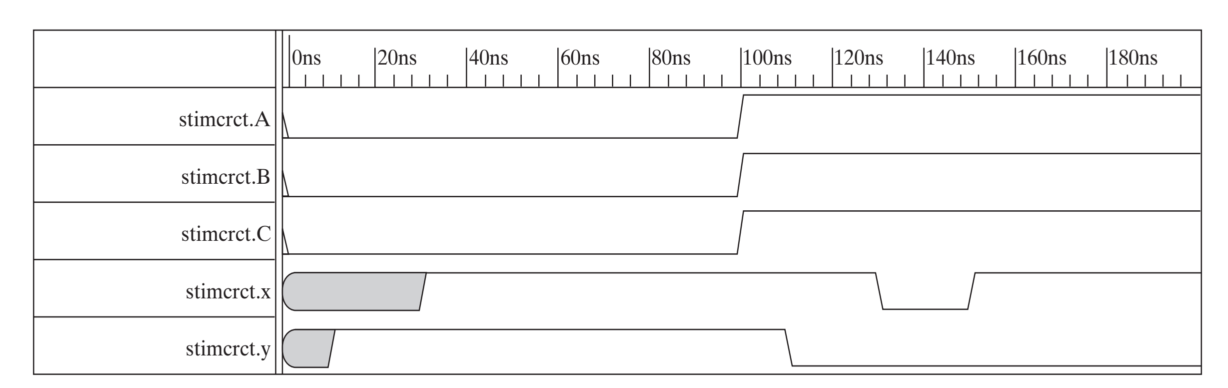


FIGURA 3-38
Salida de la simulación del ejemplo HDL 3-3

pués de 100 ns. La salida *y* se desconoce durante los primeros 10 ns, y la *x*, durante los primeros 30 ns. La salida *y* cambia de 1 a 0 a los 110 ns. La salida *x* cambia de 1 a 0 a los 130 ns y regresa a 1 a los 150 ns, tal como se predijo en la tabla 3-6.

Expresiones booleanas

Las expresiones booleanas se especifican en Verilog HDL con un enunciado de asignación continuo que consiste en la palabra clave **assign** seguida de una expresión booleana. Para distinguir el más aritmético del OR lógico, Verilog HDL usa los símbolos (&), (|) y (~) para AND, OR y NOT (complemento), respectivamente. Así pues, para describir el circuito simple de la figura 3-37 con una expresión booleana se utiliza el enunciado

```
assign x = (A & B) | ~C);
```

El ejemplo HDL 3-4 muestra la descripción de un circuito que se especifica con estas dos expresiones booleanas:

$$x = A + BC + B'D$$

$$y = B'C + BC'D'$$

Ejemplo HDL 3-4

```
//Circuito especificado con expresiones booleanas
module circuit_bln (x,y,A,B,C,D);
   input A,B,C,D;
   output x,y;
   assign x = A | (B & C) | (~B & D);
   assign y = (~B & C) | (B & ~C & ~D);
endmodule
```

El circuito tiene dos salidas, x y y, y cuatro entradas, A, B, C y D. Los dos enunciados **assign** describen las ecuaciones booleanas.

Hemos visto que es posible describir un circuito digital con enunciados HDL de la misma forma en que lo dibujamos en un diagrama de circuito, o bien especificarlo con una expresión booleana. La ventaja del HDL es que puede procesarse con una computadora.

Primitivas definidas por el usuario (UDP)

Las compuertas lógicas que usamos en las descripciones en HDL con las palabras clave **and, or,** etcétera, están definidas por el sistema y se denominan *primitivas del sistema*. El usuario puede crear más primitivas definiéndolas de forma tabular. Estos tipos de circuitos se llaman primitivas definidas por el usuario (UDP, *user-defined primitives*). Una forma de especificar un circuito digital en forma tabular es con una tabla de verdad. Las descripciones de UDP no utilizan la palabra clave **module;** se declaran con la palabra clave **primitive.** La mejor manera de explicar las declaraciones de primitivas es con un ejemplo.

El ejemplo HDL 3-5 define una UDP con una tabla de verdad. Las reglas generales son:

- Se declara con la palabra clave **primitive** seguida de un nombre y una lista de puertos.
- Sólo puede haber una salida y debe aparecer en primer lugar en la lista de puertos y declararse con la palabra clave **output.**

Ejemplo HDL 3-5

```
//Primitiva  definida por el usuario (UDP)
primitive crctp (x,A,B,C);
   output x;
   input A,B,C;
//Tabla de verdad para x(A,B,C) = Σ(0,2,4,6,7)
   table
//     A   B   C  :  x   (Esto es sólo un comentario)
       0   0   0  :  1;
       0   0   1  :  0;
       0   1   0  :  1;
       0   1   1  :  0;
       1   0   0  :  1;
       1   0   1  :  0;
       1   1   0  :  1;
       1   1   1  :  1;
   endtable
endprimitive

//Crear una copia de la primitiva
module declare_crctp;
  reg x,y,z;
  wire w;
  crctp (w,x,y,z);
endmodule
```

- Puede haber cualquier número de entradas. El orden en que aparecen en la declaración **input** debe ser el mismo en el que se les asigna valores en la tabla que sigue.
- La tabla de verdad se encierra entre las palabras clave **table** y **endtable.**
- Los valores de las entradas se dan en orden terminando con un signo de dos puntos (:). La salida siempre es el último elemento de una fila y va seguida de un punto y coma (;).
- Termina con la palabra clave **endprimitive.**

Observe que las variables que se pusieron como cabeceras de la tabla forman parte de un comentario y sólo se han incluido por claridad. El sistema reconoce las variables por el orden en que aparecen en la declaración **input.** Las primitivas definidas por el usuario se utilizan en la construcción de otros circuitos digitales igual que las primitivas del sistema. Por ejemplo, la declaración

```
crctp (w,x,y,z)
```

produce un circuito que implementa

$$w\,(x, y, z) = \Sigma(0, 2, 4, 6, 7)$$

con entradas x, y, z y salida w.

Aunque Verilog HDL utiliza este tipo de descripción únicamente para las UDP, otros HDL particulares y sistemas de diseño asistido por computadora (CAD) emplean otros procedimientos para especificar circuitos digitales en forma tabular. El software de CAD puede procesar las tablas y deducir una estructura eficiente de compuertas para el diseño.

Esta sección constituye una introducción a HDL, presentando ejemplos sencillos de modelado estructural. En el capítulo que sigue se hará una presentación más detallada de Verilog HDL. El lector que ya esté familiarizado con los circuitos combinacionales podrá pasar directamente a la sección 4-11 para continuar con este tema.

PROBLEMAS

3-1 Simplifique las siguientes funciones booleanas empleando mapas de tres variables:
a) $F(x, y, z) = \Sigma\,(0, 2, 6, 7)$
b) $F(A, B, C) = \Sigma(0, 2, 3, 4, 6)$
c) $F(a, b, c) = \Sigma\,(0, 1, 2, 3, 7)$
d) $F(x, y, z) = \Sigma S(3, 5, 6, 7)$

3-2 Simplifique las siguientes funciones booleanas empleando mapas de tres variables:
a) $F(x, y, z) = \Sigma\,(0, 1, 5, 7)$
b) $F(x, y, z) = \Sigma(1, 2, 3, 6, 7)$

3-3 Simplifique las siguientes expresiones booleanas empleando mapas de tres variables:
a) $xy + x'y'z' + x'yz'$
b) $x'y' + yz + x'yz'$
c) $A'B + BC' + B'C'$

3-4 Simplifique las siguientes funciones booleanas empleando mapas x:
a) $F(x, y, z) = \Sigma\,(2, 3, 6, 7)$
b) $F(A, B, C, D) = \Sigma(4, 6, 7, 15)$
c) $F(A, B, C, D) = \Sigma\,(3, 7, 11, 13, 14, 15)$
d) $F(w, x, y, z) = \Sigma(2, 3, 12, 13, 14, 15)$

3-5 Simplifique las siguientes funciones booleanas empleando mapas de cuatro variables:
a) $F(w, x, y, z) = \Sigma(1, 4, 5, 6, 12, 14, 15)$
b) $F(A, B, C, D) = \Sigma(0, 1, 2, 4, 5, 7, 11, 15)$
c) $F(w, x, y, z) = \Sigma(2, 3, 10, 11, 12, 13, 14, 15)$
d) $F(A, B, C, D) = \Sigma(0, 2, 4, 5, 6, 7, 8, 10, 13, 15)$

3-6 Simplifique las siguientes expresiones booleanas empleando mapas de cuatro variables:
a) $A'B'C'D' + AC'D' + B'CD' + A'BCD + BC'D$
b) $x'z + w'xy' + w(x'y + xy')$

3-7 Simplifique las siguientes expresiones booleanas empleando mapas de cuatro variables:
a) $w'z + xz + x'y + wx'z$ b) $B'D + A'BC' + AB'C + ABC'$
c) $AB'C + B'C'D' + BCD + ACD' + A'B'C + A'BC'D$
d) $wxy + yz + xy'z + x'y$

3-8 Encuentre los minitérminos de las siguientes expresiones booleanas graficando primero cada función en un mapa:
a) $xy + yz + xy'z$ b) $C'D + ABC' + ABD' + A'B'D$
c) $wxy + x'z' + w'xz$

3-9 Encuentre todos los implicantes primos de las siguientes funciones booleanas y deduzca cuáles son esenciales:
a) $F(w, x, y, z) = \Sigma(0, 2, 4, 5, 6, 7, 8, 10, 13, 15)$
b) $F(A, B, C, D) = \Sigma(0, 2, 3, 5, 7, 8, 10, 11, 14, 15)$
c) $F(A, B, C, D) = \Sigma(1, 3, 4, 5, 10, 11, 12, 13, 14, 15)$

3-10 Simplifique las siguientes funciones booleanas encontrando primero los implicantes primos esenciales:
a) $F(w, x, y, z) = \Sigma(0, 2, 4, 5, 6, 7, 8, 10, 13, 15)$
b) $F(A, B, C, D) = \Sigma(0, 2, 3, 5, 7, 8, 10, 11, 14, 15)$
c) $F(A, B, C, D) = \Sigma(1, 3, 4, 5, 10, 11, 12, 13, 14, 15)$

3-11 Simplifique las siguientes funciones booleanas empleando mapas de cinco variables:
a) $F(A, B, C, D, E) = \Sigma(0, 1, 4, 5, 16, 17, 21, 25, 29)$
b) $F = A'B'CE' + A'B'C'D' + B'D'E' + B'CD' + CDE' + BDE'$

3-12 Simplifique las siguientes funciones booleanas en forma de producto de sumas:
a) $F(w, x, y, z) = \Sigma\ (0, 2, 5, 6, 7, 8, 10)$ b) $F(A, B, C, D) = \prod(1, 3, 5, 7, 13, 15)$

3-13 Simplifique las expresiones siguientes en forma de 1) suma de productos y 2) producto de sumas:
a) $x'z' + y'z' + yz' + xy$ b) $AC' + B'D + A'CD + ABCD$
c) $(A' + B' + D')(A + B' + C')(A' + B + D')(B + C' + D')$

3-14 Dé tres posibles formas de expresar esta función booleana con ocho o menos literales:

$$F = A'B'D' + AB'CD' + A'BD + ABC'D$$

3-15 Simplifique la función booleana F, que tiene las condiciones de indiferencia d, y luego exprese la función simplificada como suma de minitérminos:
a) $F(x, y, z) = \Sigma\ (0, 1, 2, 4, 5)$
$d(x, y, z) = \Sigma\ (3, 6, 7)$
b) $F(A, B, C, D) = \Sigma(0, 6, 8, 13, 14)$
$d(A, B, C, D) = \Sigma(2, 4, 10)$
c) $F(A, B, C, D) = \Sigma(1, 3, 5, 7, 9, 15)$
$d(A, B, C, D) = \Sigma(4, 6, 12, 13)$

3-16 Simplifique estas expresiones e impleméntelas con circuitos de compuertas NAND de dos niveles:
a) $AB' + ABD + ABD' + A'C'D' + A'BC'$
b) $BD + BCD' + AB'C'D'$

3-17 Dibuje un diagrama lógico NAND que implemente el complemento de esta función:

$$F(A, B, C, D) = \Sigma(0, 1, 2, 3, 4, 8, 9, 12)$$

3-18 Dibuje un diagrama lógico empleando sólo compuertas NAND de dos entradas para implementar esta expresión:

$$(AB + A'B')(CD' + C'D)$$

3-19 Simplifique las funciones siguientes e impleméntelas con circuitos de compuertas NOR de dos niveles:

a) $F = wx' + y'z' + w'yz'$ b) $F(w, x, y, z) = \Sigma(5, 6, 9, 10)$

3-20 Dibuje el circuito NAND de múltiples niveles para esta expresión:

$$(AB' + CD')E + BC(A + B)$$

3-21 Dibuje el circuito NOR de múltiples niveles para esta expresión:

$$w(x + y + z) + xyz$$

3-22 Convierta el diagrama lógico del circuito mostrado en la figura 4-4 en un circuito NAND multinivel.

3-23 Implemente la siguiente función booleana F, que tiene las condiciones de indiferencia d, empleando no más de dos compuertas NOR:

$$F(A, B, C, D) = \Sigma(0, 1, 2, 9, 11)$$
$$d(A, B, C, D) = \Sigma(8, 10, 14, 15)$$

Suponga que cuenta con las entradas normales y con sus complementos.

3-24 Implemente la siguiente función booleana F empleando las formas de dos niveles a) NAND-AND, b) AND-NOR, c) OR-NAND, y d) NOR-OR:

$$F(A, B, C, D) = \Sigma(0, 1, 2, 3, 4, 8, 9, 12)$$

3-25 Enumere las ocho formas degeneradas de dos niveles y demuestre que se reducen a una sola operación. Explique cómo usar las formas degeneradas de dos niveles para extender el número de entradas de una compuerta.

3-26 Con la ayuda de mapas, encuentre la forma de suma de productos más simple de la función $F = fg$, donde f y g son, respectivamente,

$$f = wxy' + y'z + w'yz' + x'yz'$$

y

$$g = (w + x + y' + z')(x' + y' + z)(w' + y + z')$$

3-27 Demuestre que el dual del OR exclusivo también es su complemento.

3-28 Deduzca los circuitos de un generador de paridad de tres bits y un verificador de paridad de cuatro bits empleando un bit de paridad impar.

3-29 Implemente estas cuatro expresiones booleanas:

$$D = A \oplus B \oplus C$$
$$E = A'BC + AB'C$$
$$F = ABC' + (A' + B')C$$
$$G = ABC$$

3-30 Implemente esta expresión booleana con compuertas OR exclusivo y AND:

$$F = AB'CD' + A'BCD' + AB'C'D + A'BC'D$$

3-31 Escriba en HDL la descripción de estructura de compuertas del circuito de la figura 3-22a).

3-32 El circuito OR exclusivo de la figura 3-32a) tiene compuertas con retardo de 10 ns para los inversores, 20 ns para las compuertas AND y 30 ns para las compuertas OR. La entrada del circuito pasa de $xy = 00$ a $xy = 01$.

a) Deduzca las señales en la salida de cada compuerta desde $t = 0$ hasta $t = 50$ ns.

b) Escriba la descripción HDL del circuito, incluyendo los retardos.
c) Escriba un módulo de estímulo (similar al ejemplo HDL 3-3) y simule el circuito para verificar la respuesta de la parte a).

3-33 Escriba la descripción HDL del circuito de la figura 3-37 empleando dos expresiones booleanas.

3-34 Escriba la descripción HDL del circuito especificado por estas funciones booleanas:

$$x = A(CD + B) + BC'$$
$$y = (AB' + A'B)(C + D')$$
$$z = [(A + B)(C' + D'B)]'$$

Utilice enunciados de asignación continua.

3-35 Encuentre los errores de sintaxis en las declaraciones siguientes (tome nota de que los nombres de las compuertas primitivas son opcionales):

```
module Exmpl-3(A,B,C,D,F)
inputs A,B,C,
Output D,F;
and g1(A,B,D);
not (D,B,A);
OR (F,B,C);
endmodule;
```

3-36 Dibuje el diagrama lógico del circuito digital especificado por esta descripción HDL:

```
module circt (A,B,C,D,F);
input A,B,C,D;
output F;
wire w,x,y,z,a,d;
and (x,B,C,d);
and (y,a,C);
and (w,z,B);
or (z,y,A);
or (F,x,w);
not (a,A);
not (d,D);
endmodule
```

3-37 Una función lógica de mayoría es una función booleana que da 1 si la mayoría de las variables vale 1, y 0 en caso contrario. Escriba una primitiva definida por el usuario en HDL para una función de mayoría de tres bits.

REFERENCIAS

1. Bhasker, J. 1997. *A Verilog HDL Primer*. Allentown, PA: Star Galaxy Press.
2. Hill, F. J. y G. R. Peterson. 1981. *Introduction to Switching Theory and Logical Design*, 3a. ed. Nueva York: John Wiley.

3. *IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language* (Norma IEEB 1364-1995). 1995. Nueva York: The Institute of Electrical and Electronics Engineers.
4. KARNAUGH, M. A Map Method for Synthesis of Combinational Logic Circuits. *Transactions of AIEE, Communication and Electronics*. 72, parte I (nov. de 1953): 593-99.
5. KOHAVI, Z. 1978. *Switching and Automata Theory*, 2a. ed. Nueva York: McGraw-Hill.
6. MANO, M. M. y C. R. KIME. 2000. *Logic and Computer Design Fundamentals*, 2a. ed. Upper Saddle River, NJ: Prentice-Hall.
7. MCCLUSKEY, E. J. 1986. *Logic Design Principles*. Englewood Cliffs, NJ: Prentice-Hall.
8. PALNITKAR. S. 1996. *Verilog HDL: A Guide to Digital Design and Synthesis*. SunSoft Press (un título Prentice-Hall).

4 Lógica combinacional

4-1 CIRCUITOS COMBINACIONALES

Los circuitos lógicos para sistemas digitales pueden ser combinacionales o secuenciales. Un circuito combinacional consiste en compuertas lógicas cuyas salidas en cualquier momento están determinadas por la combinación actual de entradas. Un circuito combinacional realiza una operación que se puede especificar lógicamente con un conjunto de funciones booleanas. Los circuitos secuenciales usan elementos de almacenamiento además de compuertas lógicas, y sus salidas son función de las entradas y del estado de los elementos de almacenamiento. Esto último, a su vez, es función de entradas anteriores. Por ello, las salidas de un circuito secuencial dependen no sólo de los valores actuales de las entradas, sino también de entradas anteriores, y el comportamiento del circuito se debe especificar con una sucesión temporal de entradas y estados internos. Los circuitos secuenciales se estudiarán en los capítulos 5 y 9.

Un circuito combinacional consiste en variables de entrada, compuertas lógicas y variables de salida. Las compuertas lógicas aceptan señales de las entradas y generan señales para las salidas. Este proceso transforma información binaria, de los datos de entrada dados a los datos de salida requeridos. En la figura 4-1 se presenta un diagrama de bloques de un circuito combinacional. Las n variables binarias de entrada provienen de una fuente externa; las m variables de salida van a un destino externo. Cada variable de entrada y de salida existe físicamente como una señal binaria que representa 1 lógico y 0 lógico. En muchas aplicaciones, el origen

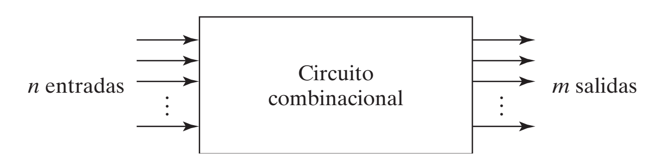


FIGURA 4-1
Diagrama de bloques de un circuito combinacional

y el destino son registros de almacenamiento. Si los registros se incluyen con las puertas combinacionales, el circuito total se considera como un circuito secuencial.

Con n variables de entrada, hay 2^n posibles combinaciones de entradas binarias. Para cada una de esas combinaciones, hay un posible valor de salida. Por tanto, es posible especificar un circuito combinacional con una tabla de verdad que presenta los valores de salida para cada combinación de variables de entrada. También es factible describir un circuito combinacional con m funciones booleanas, una para cada variable de salida. Cada función de salida se expresa en términos de las n variables de entrada.

En el capítulo 1 se estudiaron los números binarios y los códigos binarios que representan cantidades discretas de información. Las variables binarias se representan físicamente con voltajes eléctricos o algún otro tipo de señal. Las señales se pueden manipular en compuertas lógicas digitales para efectuar las funciones requeridas. En el capítulo 2 se definió el álgebra booleana como una forma de expresar las funciones lógicas algebraicamente. En el capítulo 3 se explicó la manera de simplificar las funciones booleanas para lograr implementaciones económicas con compuertas. El propósito del presente capítulo es utilizar los conocimientos adquiridos en capítulos anteriores y formular procedimientos sistemáticos para el análisis y diseño de circuitos combinacionales. La resolución de algunos ejemplos representativos proporcionará un catálogo útil de funciones elementales importantes para entender los sistemas digitales.

Hay varios circuitos combinacionales que se usan ampliamente en el diseño de sistemas digitales. Esos circuitos pueden conseguirse en circuitos integrados y se clasifican como componentes estándar. Efectúan funciones digitales específicas que se necesitan a menudo en el diseño de sistemas digitales. En este capítulo presentaremos los circuitos combinacionales estándar más importantes, como los sumadores, restadores, comparadores, decodificadores, codificadores y multiplexores. Estos componentes se fabrican como circuitos MSI (de integración a mediana escala), y también se usan como *celdas estándar* en circuitos VLSI complejos como los circuitos integrados para aplicaciones específicas (ASIC). Las funciones de la celda estándar se interconectan dentro del circuito VLSI del mismo modo que se usan en el diseño MSI de múltiples CI.

4-2 PROCEDIMIENTO DE ANÁLISIS

El análisis de un circuito combinacional requiere deducir la función que realiza el circuito. Este proceso parte de un diagrama lógico dado y culmina en un conjunto de funciones booleanas, una tabla de verdad o una posible explicación del funcionamiento del circuito. Si el diagrama lógico a analizar va acompañado de un nombre de función o de una explicación de lo que se supone que hace, el problema de análisis se reducirá a una verificación de la función planteada. El análisis se efectúa manualmente encontrando las funciones booleanas o la tabla de verdad, o bien, utilizando un programa de simulación en computadora.

El primer paso del análisis consiste en asegurarse de que el circuito dado sea combinacional y no secuencial. El diagrama de un circuito combinacional tiene compuertas lógicas sin trayectorias de retroalimentación ni elementos de memoria. Una trayectoria de retroalimentación es una conexión de la salida de una compuerta a la entrada de una segunda compuerta que forma parte de la entrada a la primera compuerta. Las trayectorias de retroalimentación en un circuito digital definen a un circuito secuencial y deben analizarse según los procedimientos delineados en el capítulo 9.

Una vez que se verifica que el diagrama lógico representa un circuito combinacional, se procede a obtener las funciones booleanas de salida o la tabla de verdad. Si se está investigan-

do la función del circuito, será necesario interpretar la operación de éste a partir de las funciones booleanas o la tabla de verdad obtenidas. El éxito de tal investigación será más asequible si tenemos experiencia previa con una amplia variedad de circuitos digitales.

Para obtener las funciones booleanas de salida a partir de un diagrama lógico, el procedimiento es el siguiente:

1. Rotule con símbolos arbitrarios todas las salidas de compuerta que son función de variables de entrada. Determine las funciones booleanas para cada salida de compuerta.
2. Rotule con otros símbolos arbitrarios las compuertas que son función de variables de entrada y de compuertas previamente rotuladas. Obtenga las funciones booleanas de estas compuertas.
3. Repita el proceso bosquejado en el paso 2 hasta obtener las salidas del circuito.
4. Por sustitución repetida de funciones previamente definidas, obtenga las funciones booleanas de salida en términos de variables de entrada.

El análisis del circuito combinacional de la figura 4-2 ilustra el procedimiento propuesto. Observe que el circuito tiene tres entradas binarias —A, B y C— y dos salidas binarias —F_1 y F_2. Las salidas de diversas compuertas están rotuladas con símbolos intermedios. Las salidas de compuertas que son función únicamente de variables de entrada son T_1 y T_2. La salida F_2 se deduce fácilmente de las variables de entrada. Las funciones booleanas de estas tres salidas son:

$$F_2 = AB + AC + BC$$
$$T_1 = A + B + C$$
$$T_2 = ABC$$

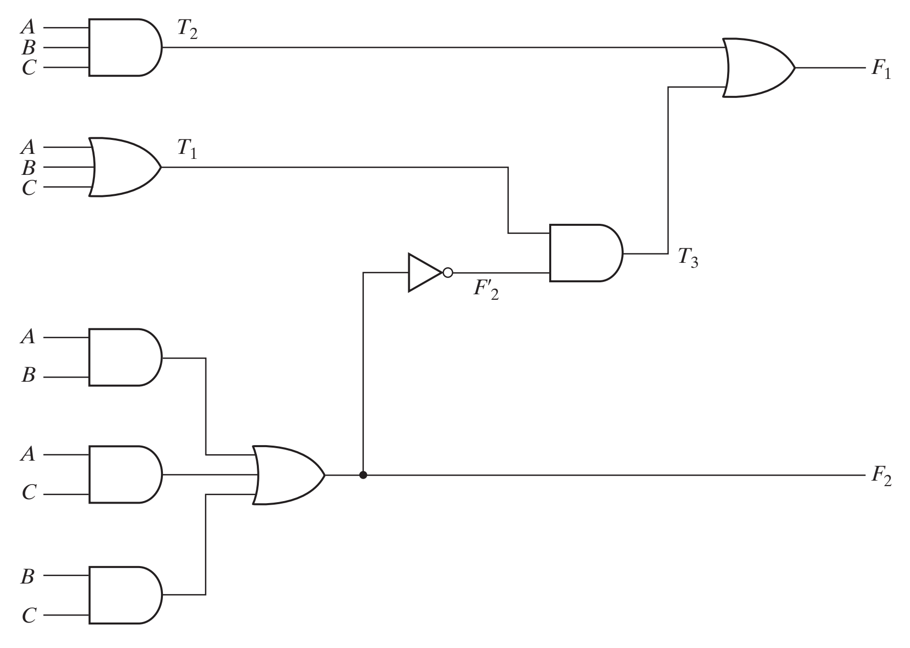


FIGURA 4-2
Diagrama lógico para el ejemplo de análisis

A continuación, consideramos las salidas de compuertas que son función de símbolos ya definidos:

$$T_3 = F'_2 T_1$$
$$F_1 = T_3 + T_2$$

Para obtener F_1 en función de A, B y C, se realiza la siguiente serie de sustituciones:

$$\begin{aligned} F_1 &= T_3 + T_2 = F'_2 T_1 + ABC = (AB + AC + BC)'(A + B + C) + ABC \\ &= (A' + B')(A' + C')(B' + C')(A + B + C) + ABC \\ &= (A' + B'C')(AB' + AC' + BC' + B'C) + ABC \\ &= A'BC' + A'B'C + AB'C' + ABC \end{aligned}$$

Si se quiere investigar más a fondo y deducir la tarea de transformación de información que este circuito efectúa, habrá que dibujar el circuito a partir de las expresiones booleanas obtenidas y tratar de reconocer una operación conocida. Las funciones booleanas para F_1 y F_2 implementan el circuito que se muestra en la figura 4-7 (sección 4-4) y equivalen a un circuito sumador completo.

La deducción de la tabla de verdad del circuito es un proceso sencillo una vez que se conocen las funciones booleanas de salida. Para obtener la tabla de verdad directamente del diagrama lógico sin tener que deducir las funciones booleanas, se procede así:

1. Determine el número de variables de entrada del circuito. Para n entradas, forme las 2^n posibles combinaciones y haga una lista de los números binarios de 0 a $2^n - 1$ en una tabla.
2. Rotule las salidas de compuertas selectas con símbolos arbitrarios.
3. Obtenga la tabla de verdad para las salidas de aquellas compuertas que son función únicamente de las variables de entrada.
4. Obtenga la tabla de verdad para las salidas de aquellas compuertas que son función de valores previamente definidos, hasta llenar las columnas de todas las salidas.

Este proceso se ilustra empleando el circuito de la figura 4-2. En la tabla 4-1, formamos las ocho posibles combinaciones de las tres variables de entrada. La tabla de verdad para F_2 se de-

Tabla 4-1
Tabla de verdad para el diagrama lógico de la figura 4-2

A	B	C	F_2	F'_2	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

termina directamente de los valores de A, B y C, siendo F_2 igual a 1 para cualquier combinación que tiene dos o tres entradas iguales a 1. La tabla de verdad para F'_2 es el complemento de F_2. Las tablas de verdad para T_1 y T_2 son las funciones OR y AND de las variables de entrada, respectivamente. Los valores para T_3 se deducen de T_1 y F'_2: T_3 es igual a 1 cuando tanto T_1 como F'_2 son 1, e igual a 0 en los demás casos. Por último, F_1 es igual a 1 para aquellas combinaciones en las que T_2 o T_3, o ambas, son 1. Una inspección de las combinaciones de A, B, C, F_1 y F_2 en la tabla de verdad revela que es idéntica a la tabla de verdad del sumador completo que se da en la sección 4-4 para x, y, z, S y C, respectivamente.

Otra forma de analizar un circuito combinacional es efectuando simulación lógica. En la sección 4-11 ilustraremos la simulación lógica y verificación del circuito de la figura 4-2 empleando Verilog HDL. (Véase el ejemplo HDL 4-10.)

4-3 PROCEDIMIENTO DE DISEÑO

El diseño de circuitos combinacionales parte de la especificación del problema y culmina en un diagrama lógico de circuitos o un conjunto de funciones booleanas a partir de las cuales se puede obtener el diagrama lógico. El procedimiento implica los pasos siguientes:

1. De las especificaciones del circuito, deduzca el número requerido de entradas y salidas; asigne un símbolo a cada una.
2. Deduzca la tabla de verdad que define la relación requerida entre las entradas y las salidas.
3. Obtenga las funciones booleanas simplificadas para cada salida en función de las variables de entrada.
4. Dibuje el diagrama lógico y verifique que el diseño sea correcto.

La tabla de verdad de un circuito combinacional consta de columnas de entrada y columnas de salida. Las columnas de entrada se obtienen de los 2^n números binarios para las n variables de entrada. Los valores binarios de las salidas se deducen de las especificaciones planteadas. Las funciones de salida especificadas en la tabla de verdad dan la definición exacta del circuito combinacional. Es importante interpretar correctamente las especificaciones verbales en la tabla de verdad. Tales especificaciones suelen ser incompletas, y cualquier interpretación errónea podría dar pie a una tabla de verdad incorrecta.

Las funciones binarias de salida enumeradas en la tabla de verdad se simplifican con cualquier método disponible, como manipulación algebraica, el método de mapa o un programa de simplificación para computadora. En muchos casos habrá diversas expresiones simplificadas para escoger. En cada aplicación dada, ciertos criterios servirán como guía para escoger una implementación. Un diseño práctico debe tomar en cuenta restricciones como el número de compuertas, el número de entradas de una compuerta, el tiempo de propagación de la señal a través de las compuertas, el número de interconexiones, las limitaciones de la corriente que proporciona cada compuerta y diversos criterios adicionales que es preciso considerar al diseñar con circuitos integrados. Puesto que la importancia de cada restricción depende de la aplicación específica, es difícil hacer recomendaciones generales acerca de lo que constituye una implementación aceptable. En la mayoría de los casos, la simplificación comienza por satisfacer un objetivo elemental, como producir las funciones booleanas simplificadas en una forma estándar, y luego efectúa otros pasos para cumplir con otros criterios de desempeño.

Ejemplo de conversión de código

La disponibilidad de una gran variedad de códigos para los mismos elementos discretos de información hace que diferentes sistemas digitales usen códigos distintos. A veces es necesario usar la salida de un sistema como entrada de otro, y hay que insertar un circuito de conversión entre los dos sistemas si cada uno usa un código distinto para la misma información. Así pues, un convertidor de código es un circuito que hace compatibles a los dos sistemas aunque cada uno utilice un código binario distinto.

Para convertir del código binario A al código binario B, las líneas de entrada deberán proporcionar la combinación de elementos que especifica el código A, y las líneas de salida deberán generar las combinaciones de bits correspondientes del código B. Un circuito combinacional efectúa esta transformación con compuertas lógicas. Ilustraremos el procedimiento de diseño con un ejemplo que convierte el código BCD (decimal codificado en binario) en código exceso-3 para los dígitos decimales.

Las combinaciones de bits asignadas a los códigos BCD y exceso-3 se incluyen en la tabla 1-5 (sección 1-7). Puesto que ambos códigos usan cuatro bits para representar un dígito decimal, deberá haber cuatro variables de entrada y cuatro variables de salida. Designaremos a las primeras con A, B, C y D, y a las variables de salida, con w, x, y y z. La tabla de verdad que relaciona las variables de entrada y de salida se presenta en la tabla 4-2. Las combinaciones de bits para las entradas y sus salidas correspondientes se obtienen directamente de la sección 1-7. Cabe señalar que cuatro variables binarias pueden tener 16 combinaciones de bits, pero sólo 10 de ellas se presentan en la tabla de verdad. Las otras seis son combinaciones indiferentes. Esos valores carecen de significado en BCD y suponemos que nunca se presentarán. Por tanto, es posible asignar a las variables de salida 1 o 0, lo que produzca un circuito más simple.

Se han trazado los mapas de la figura 4-3 a fin de obtener funciones booleanas simplificadas para las salidas. Cada mapa representa una de las cuatro salidas del circuito en función de las cuatro variables de entrada. Los unos dentro de los cuadrados se obtienen de los minitér-

Tabla 4-2
Tabla de verdad para el ejemplo de conversión de código

Entrada BCD				Salida código exceso-3			
A	*B*	*C*	*D*	*w*	*x*	*y*	*z*
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

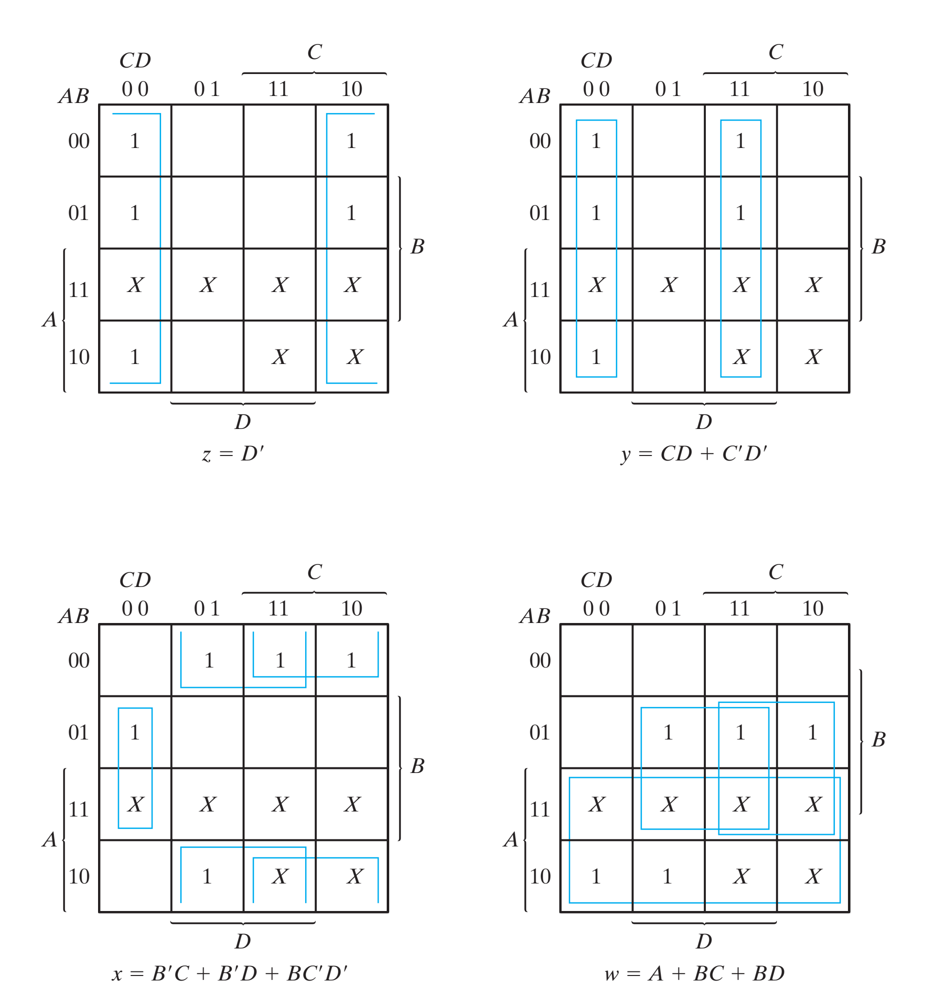


FIGURA 4-3
Mapas para el convertidor de código BCD a exceso-3

minos que hacen que la salida sea 1, y se obtienen de la tabla de verdad examinando las columnas de salida una por una. Por ejemplo, la columna de la salida z tiene cinco unos; por tanto, el mapa de z tiene cinco unos, cada uno en el cuadrado correspondiente al minitérmino que hace que z sea 1. Los seis minitérminos indiferentes, del 10 al 15, se han marcado con X. Bajo el mapa de cada variable se da una posible forma de simplificar la función en forma de suma de productos.

Es posible obtener un diagrama lógico de dos niveles directamente de las expresiones booleanas deducidas de los mapas. Hay otras posibilidades para un diagrama lógico que implemente este circuito. Las expresiones obtenidas en la figura 4-3 podrían manipularse algebraicamente con el fin de usar compuertas comunes para dos o más salidas. Esta manipu-

lación que se presenta a continuación, ilustra la flexibilidad que se obtiene con sistemas de múltiples salidas implementados con tres o más niveles de compuertas:

$$z = D'$$
$$y = CD + C'D' = CD + (C + D)'$$
$$x = B'C + B'D + BC'D' = B'(C + D) + BC'D'$$
$$= B'(C + D) + B(C + D)'$$
$$w = A + BC + BD = A + B(C + D)$$

El diagrama lógico que implementa estas expresiones aparece en la figura 4-4. Observe que se ha usado la compuerta OR cuya salida es $C + D$ para implementar parcialmente cada una de las tres salidas.

Sin contar los inversores de entradas, la implementación en forma de suma de productos requiere siete compuertas AND y tres compuertas OR. La implementación de la figura 4-4 requiere cuatro compuertas AND, cuatro compuertas OR y un inversor. Si sólo se cuenta con las entradas normales, la primera implementación requerirá inversores para las variables B, C y D, y la segunda, para las variables B y D.

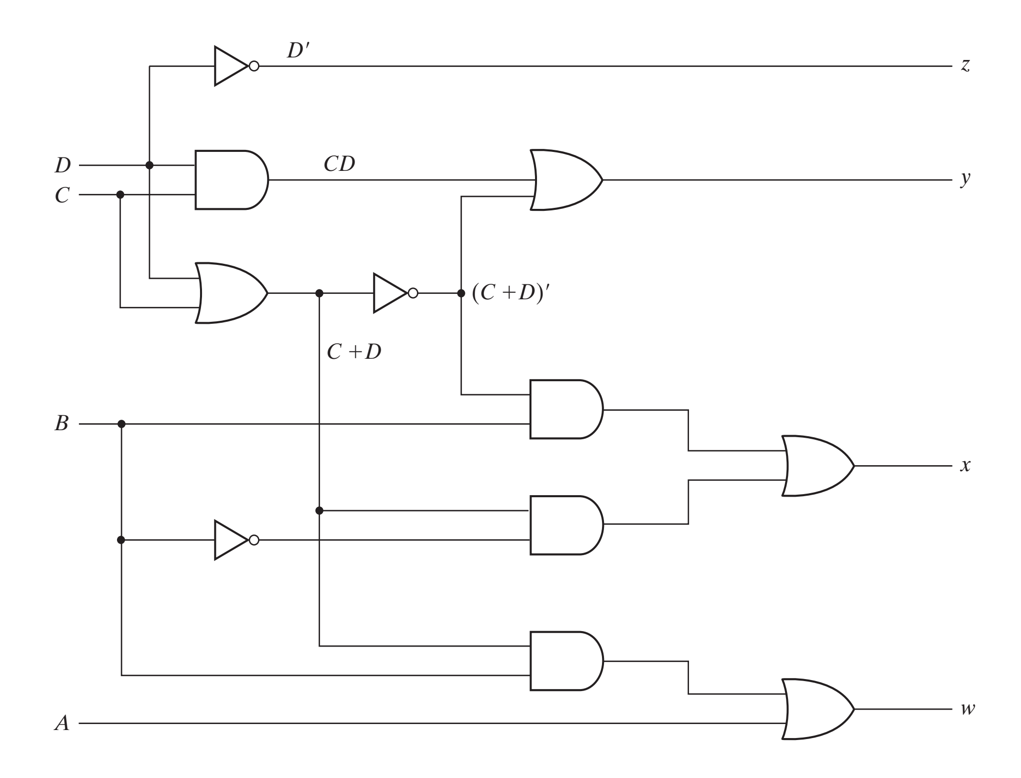


FIGURA 4-4
Diagrama lógico para el convertidor de código BCD a exceso-3

4-4 SUMADOR-RESTADOR BINARIO

Las computadoras digitales efectúan diversas tareas de procesamiento de información. Entre esas funciones están las operaciones aritméticas. La operación aritmética más básica es la suma de dos dígitos binarios. Esta suma simple consiste en cuatro posibles operaciones elementales: $0 + 0 = 0$, $0 + 1 = 1$, $1 + 0 = 1$ y $1 + 1 = 10$. Las primeras tres operaciones producen una suma de un dígito, pero cuando ambos bits sumandos son 1, la suma binaria consta de dos dígitos. El bit más significativo de este resultado se denomina *acarreo* (*carry*, en inglés). Cuando ambos sumandos contienen más dígitos significativos, el acarreo obtenido de la suma de dos bits se suma al siguiente par más alto de bits significativos. Un circuito combinacional que realiza la suma de dos bits se denomina *semisumador;* uno que realiza la suma de tres bits (dos bits significativos y un acarreo previo) es un *sumador completo*. Los nombres de los circuitos provienen del hecho de que es posible usar dos semisumadores para implementar un sumador completo.

Un sumador-restador binario es un circuito combinacional que realiza las operaciones aritméticas de suma y resta con números binarios. Desarrollaremos este circuito utilizando un diseño jérárquico. Primero diseñaremos el semisumador, y a partir de él desarrollaremos el sumador completo. La conexión de n sumadores completos en cascada produce un sumador binario para números de n bits. Incluiremos el circuito de resta con la ayuda de un circuito complementador.

Semisumador

Por la descripción verbal del semisumador, se sabe que este circuito necesita dos entradas binarias y dos salidas binarias. Las variables de entrada designan los bits sumandos; las de salida, la suma y el acarreo. Asignaremos los símbolos x y y a las dos entradas y S (de suma) y C (de *carry*) a las salidas. La tabla de verdad del semisumador se presenta en la tabla 4-3. La salida C es 1 sólo cuando ambas entradas son 1. La salida S representa el bit menos significativo de la suma.

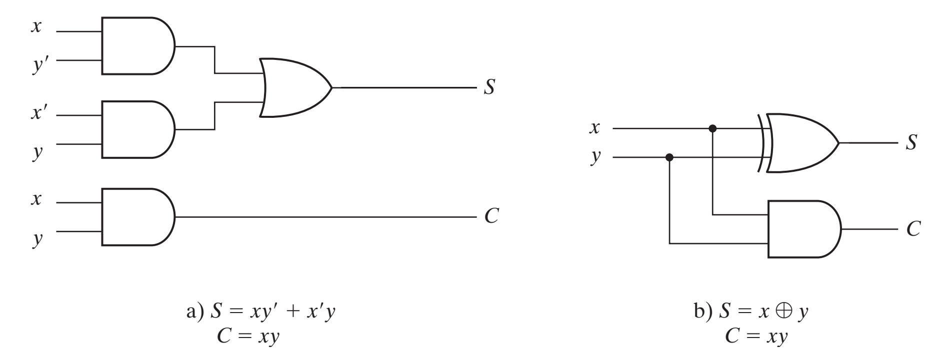
Las funciones booleanas simplificadas para las dos salidas se obtienen directamente de la tabla de verdad. Las expresiones simplificadas en suma de productos son

$$S = x'y + xy'$$
$$C = xy$$

El diagrama lógico del semisumador implementado como suma de productos se observa en la figura 4-5a). También se puede implementar con un OR exclusivo y una compuerta AND, como se indica en la figura 4-5b). Esta forma se utiliza para mostrar cómo dos semisumadores sirven para construir un sumador completo.

Tabla 4-3
Semisumador

x	*y*	*C*	*S*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



a) $S = xy' + x'y$
$C = xy$

b) $S = x \oplus y$
$C = xy$

FIGURA 4-5
Implementación de semisumador

Sumador completo

Un sumador completo es un circuito combinacional que forma la suma aritmética de tres bits. Tiene tres entradas y dos salidas. Dos de las variables de entrada, denotadas por x y y, representan los dos bits significativos que se sumarán. La tercera entrada, z, representa el acarreo de la posición significativa inmediata inferior. Se requieren dos salidas porque la suma aritmética de tres dígitos binarios puede tener valores entre 0 y 3, y el 2 o el 3 binarios requieren dos dígitos. Las dos salidas se designan otra vez con los símbolos S y C. La variable binaria S da el valor del bit menos significativo de la suma. La variable binaria C da el acarreo de salida. La tabla de verdad del sumador completo se presenta en la tabla 4-4. Las ocho filas bajo las variables de entrada dan todas las posibles combinaciones de las tres variables. Las variables de salida se determinan a partir de la suma aritmética de los bits de entrada. Si todos los bits de entrada son 0, la salida es 0. La salida S es 1 cuando sólo una entrada es 1 o cuando las tres entradas son 1. La salida C da un acarreo de 1 si dos o tres entradas son 1.

Los bits de entrada y de salida del circuito combinacional tienen diferentes interpretaciones en las distintas etapas del problema. Físicamente, las señales binarias de las entradas se consideran dígitos binarios que deben sumarse aritméticamente para formar una salida de dos dígitos. Por otra parte, los mismos valores binarios se consideran variables de funciones booleanas cuando se expresan en la tabla de verdad o cuando el circuito se implementa con compuertas

Tabla 4-4
Sumador completo

x	*y*	*z*	*C*	*S*
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

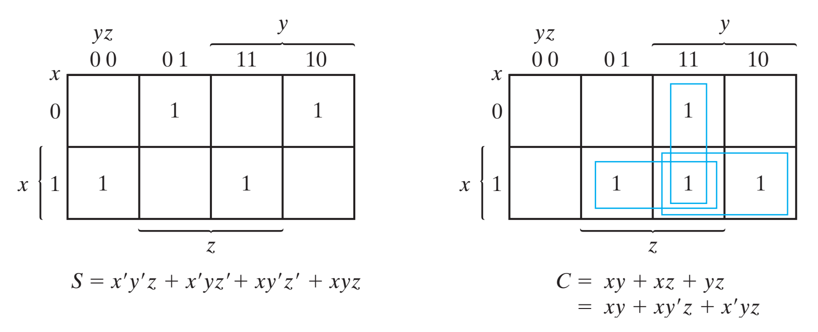


FIGURA 4-6
Mapas para el sumador completo

lógicas. Los mapas para las salidas del sumador completo aparecen en la figura 4-6. Las expresiones simplificadas son

$$S = x'y'z + x'yz' + xy'z' + xyz$$
$$C = xy + xz + yz$$

El diagrama lógico para el sumador completo implementado en forma de suma de productos se muestra en la figura 4-7. También puede implementarse con dos semisumadores y una compuerta OR, como se indica en la figura 4-8. La salida S del segundo semisumador es el OR exclusivo de z y la salida del primer semisumador, lo que da

$$\begin{aligned} S &= z \oplus (x \oplus y) \\ &= z'(xy' + x'y) + z(xy' + x'y)' \\ &= z'(xy' + x'y) + z(xy + x'y') \\ &= xy'z' + x'yz' + xyz + x'y'z \end{aligned}$$

La salida de acarreo es

$$C = z(xy' + x'y) + xy = xy'z + x'yz + xy$$

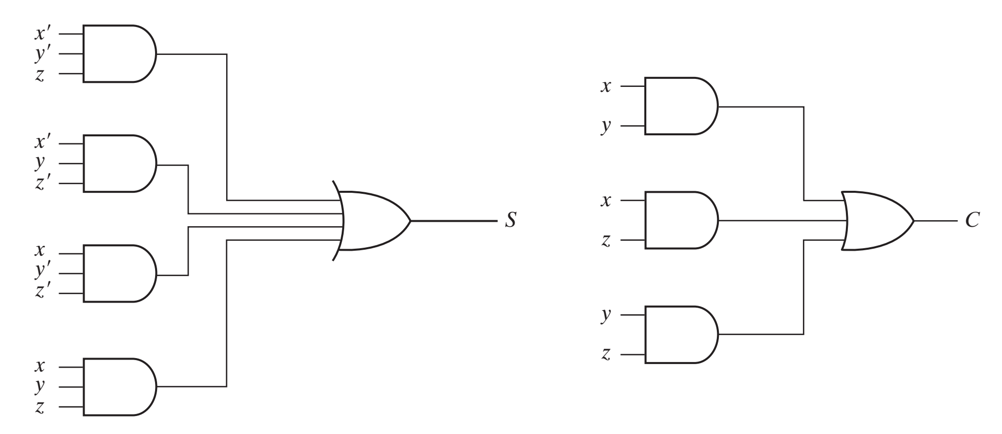


FIGURA 4-7
Implementación de un sumador completo como suma de productos

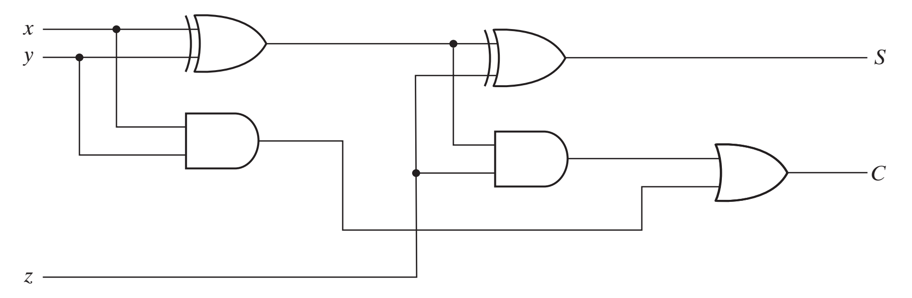


FIGURA 4-8
Implementación de un sumador completo con dos semisumadores y una compuerta OR

Sumador binario

Un sumador binario es un circuito digital que produce la suma aritmética de dos números binarios. Es posible construirlo con sumadores completos dispuestos en cascada, conectando el acarreo de salida de cada sumador completo al acarreo de entrada del siguiente sumador completo de la cadena. La figura 4-9 muestra la interconexión de cuatro circuitos sumadores completos (SC) para formar un sumador binario de cuatro bits con acarreo rizado. Los bits de los sumandos A y B se designan con subíndices de izquierda a derecha; el subíndice 0 denota el bit menos significativo. Los acarreos se conectan en una cadena a través de los sumadores completos. El acarreo de entrada del sumador es C_0 y se propaga a través de los sumadores completos hasta el acarreo de salida C_4. Las salidas S generan los bits de suma requeridos. Un sumador de n bits requiere n sumadores completos con cada acarreo de salida conectado al acarreo de entrada del siguiente sumador completo de orden superior.

Para ilustrar esto con un ejemplo específico, consideremos los dos números binarios $A = 1011$ y $B = 0011$. Su suma $S = 1110$ se forma con el sumador de cuatro bits así:

Subíndice i:	**3**	**2**	**1**	**0**	
Acarreo de entrada	0	1	1	0	C_i
Sumando	1	0	1	1	A_i
Sumando	0	0	1	1	B_i
Suma	1	1	1	0	S_i
Acarreo de salida	0	0	1	1	C_{i+1}

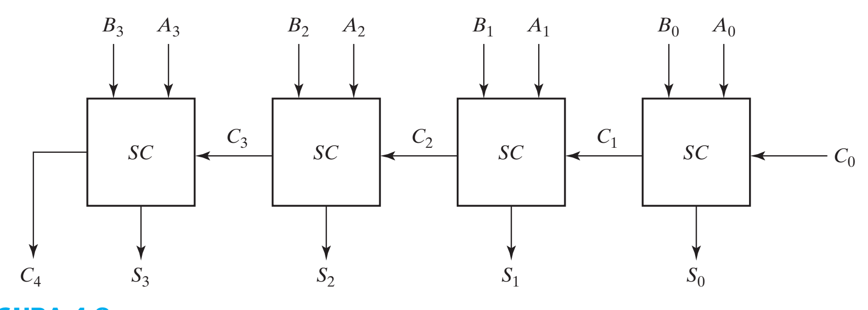


FIGURA 4-9
Sumador de cuatro bits

Los bits se suman con sumadores completos, comenzando por la posición menos significativa (subíndice 0) para formar el bit de suma y el bit de acarreo. El acarreo de entrada C_0 en la posición menos significativa debe ser 0. El valor de C_{i+1} en una posición significativa dada es el acarreo de salida del sumador completo. Este valor se transfiere al acarreo de entrada del sumador completo que suma los bits de la siguiente posición significativa a la izquierda. Así, los bits de la suma se generan comenzando por la posición de la extrema derecha y están disponibles tan pronto como se genera el bit de acarreo anterior. Se deben generar todos los acarreos para que los bits de suma correctos aparezcan en las salidas.

El sumador de cuatro bits es un ejemplo representativo de un componente estándar. Se utiliza en muchas aplicaciones que implican operaciones aritméticas. Observe que el diseño de este circuito empleando el método clásico requeriría una tabla de verdad con $2^9 = 512$ entradas, ya que el circuito tiene nueve entradas. Al usar un método iterativo de conectar en cascada una función estándar, es posible obtener una implementación sencilla y directa.

Propagación del acarreo

La suma de dos números binarios en paralelo implica que todos los bits de los sumandos están disponibles al mismo tiempo para efectuar el cálculo. Como en cualquier circuito combinacional, la señal tiene que propagarse a través de las compuertas para que la salida correcta (la suma) esté disponible en las terminales de salida. El tiempo total de propagación es igual al retardo de propagación de una compuerta representativa multiplicado por el número de niveles de compuertas del circuito. El retardo de propagación más largo en un sumador es el tiempo que el acarreo tarda en propagarse a través de los sumadores completos. Dado que cada bit de la suma depende del valor del acarreo de entrada, el valor de S_i en cualquier etapa dada del sumador alcanzará su valor final de estado estable sólo hasta que el acarreo de entrada se haya propagado a esa etapa. Consideremos la salida S_3 de la figura 4-9. Las entradas A_3 y B_3 están disponibles tan pronto como se aplican señales de entrada al sumador. Sin embargo, el acarreo de entrada C_3 no se estabiliza en su valor final sino hasta después de que se cuenta con C_2 de la etapa anterior. Asimismo, C_2 tiene que esperar a C_1, y así hasta C_0. Por tanto, no será sino hasta que el acarreo se propague en rizo a través de todas las etapas cuando la última salida S_3 y el último acarreo C_4 se estabilicen en su valor final correcto.

Es posible calcular el número de niveles de compuerta para la propagación del acarreo a partir del circuito del sumador completo. La figura 4-10 reproduce otra vez el circuito. Las variables de entrada y salida llevan el subíndice i para denotar una etapa representativa del

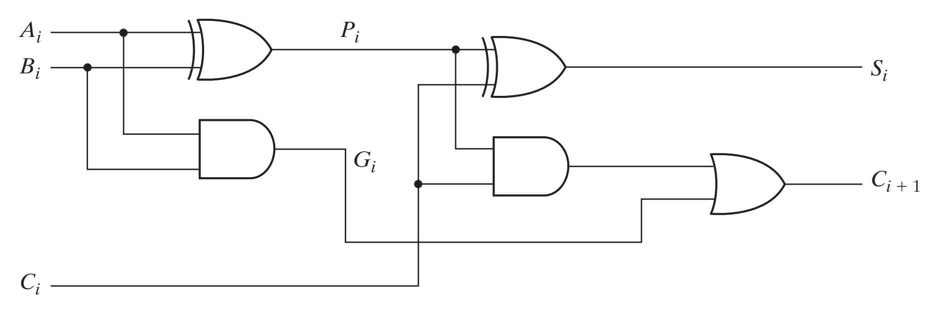


FIGURA 4-10
Sumador completo en el que se indican P y G

sumador. Las señales en P_i y G_i se estabilizan en sus valores de estado estable después de que se propagan a través de sus respectivas compuertas. Todos los sumadores completos tienen en común estas dos señales y sólo dependen de los bits de entrada de los sumandos. La señal del acarreo de entrada C_i al acarreo de salida C_{i+1} se propaga a través de una compuerta AND y una OR, lo que constituye dos niveles de compuertas. Si hay cuatro sumadores completos en el sumador, el acarreo de salida C_4 tendrá $2 \times 4 = 8$ niveles de compuerta desde C_0 hasta C_4. En el caso de un sumador de n bits, el acarreo tendrá que propagarse a través de $2n$ niveles de compuertas desde la entrada hasta la salida.

El tiempo de propagación del acarreo es un factor que limita la rapidez con que se suman dos números. Aunque el sumador, o cualquier circuito combinacional, siempre tendrá algún valor en sus terminales de salida, esos valores no serán correctos si no se da a las señales el tiempo suficiente para propagarse a través de las compuertas conectadas entre las entradas y las salidas. Puesto que todas las demás operaciones aritméticas se implementan con sumas sucesivas, el tiempo consumido durante el proceso de adición es crucial. Una solución obvia para reducir el retardo de propagación del acarreo es utilizar compuertas más rápidas con menor retardo. Sin embargo, los circuitos físicos tienen un límite en este sentido. Otra solución sería aumentar la complejidad del equipo de modo tal que el retardo del acarreo se reduzca. Existen varias técnicas para reducir el tiempo de propagación del acarreo en un sumador paralelo. La técnica más ampliamente utilizada se vale del principio de *acarreo anticipado*.

Considere el circuito del sumador completo que se aprecia en la figura 4-10. Si definimos dos nuevas variables binarias

$$P_i = A_i \oplus B_i$$
$$G_i = A_i B_i$$

la suma y el acarreo se expresarán así:

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_i C_i$$

G_i se llama *acarreo generado* y produce un acarreo de 1 si tanto A_i como B_i son 1, independientemente del acarreo de entrada C_i. P_i se llama *acarreo propagado* porque es el término asociado a la propagación del acarreo de C_i a C_{i+1}.

Ahora escribiremos las funciones booleanas para los acarreos de salida de cada etapa y sustituiremos C_i por el valor obtenido de las ecuaciones anteriores:

$$C_0 = \text{acarreo de entrada}$$
$$C_1 = G_0 + P_0 C_0$$
$$C_2 = G_1 + P_1 C_1 = G_1 + P_1(G_0 + P_0 C_0) = G_1 + P_1 G_0 + P_1 P_0 C_0$$
$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$$

Puesto que la función booleana para cada acarreo de salida se expresa en forma de suma de productos, cada función se puede implementar con un nivel de compuertas AND seguido de una compuerta OR (o con dos niveles de NAND). Las tres funciones booleanas de C_1, C_2 y C_3 se

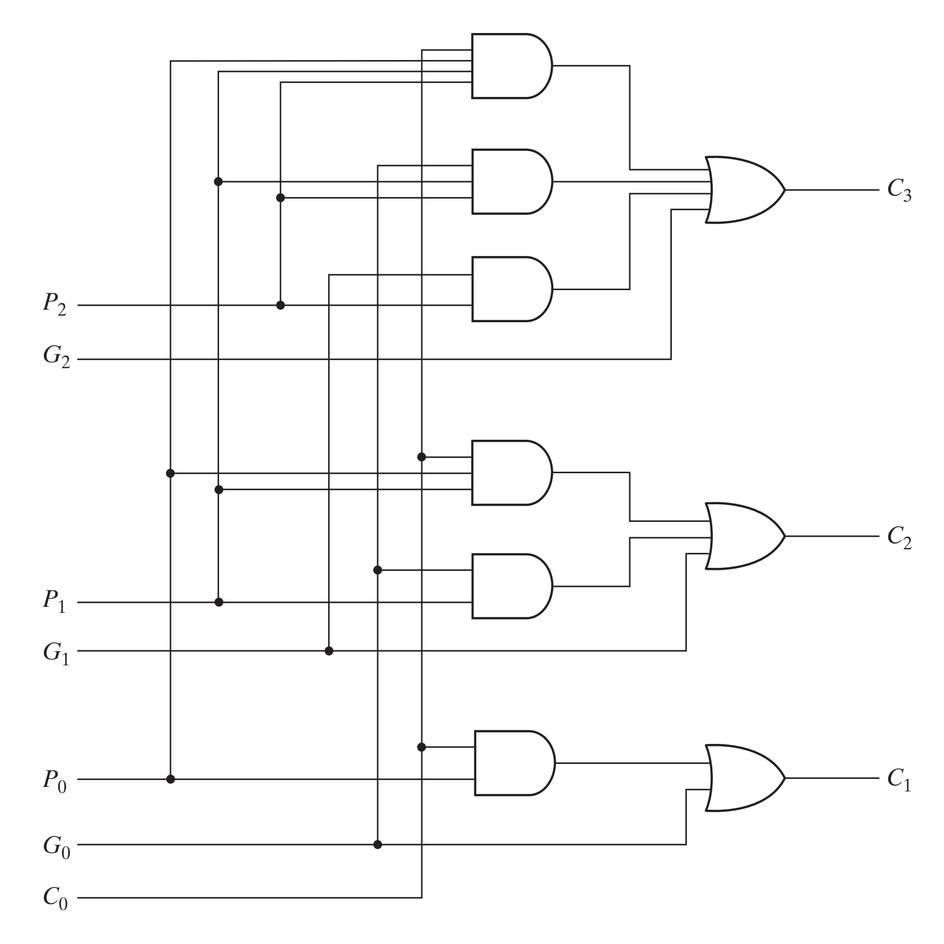


FIGURA 4-11
Diagrama lógico del generador de acarreo anticipado

implementan en el generador de acarreo anticipado que se observa en la figura 4-11. Advierta que C_3 no tiene que esperar a que C_2 y C_1 se propaguen; de hecho, C_3 se propaga al mismo tiempo que C_1 y C_2.

La construcción de un sumador de cuatro bits con un esquema de acarreo anticipado se representa en la figura 4-12. Cada salida de suma requiere dos compuertas OR exclusivo. La salida de la primera compuerta OR exclusivo genera la variable P_i y la compuerta AND genera la variable G_i. Los acarreos se propagan mediante el generador de acarreo anticipado (similar al de la figura 4-11) y se aplican como entradas a la segunda compuerta OR exclusivo. Todos los acarreos de salida se generan después de un retardo de dos niveles de compuertas. Así, las salidas S_1 a S_3 tienen el mismo tiempo de retardo por propagación. No se muestra el circuito de dos niveles para el acarreo de salida C_4. Este circuito se deduce fácilmente empleando el método de sustitución de ecuaciones.

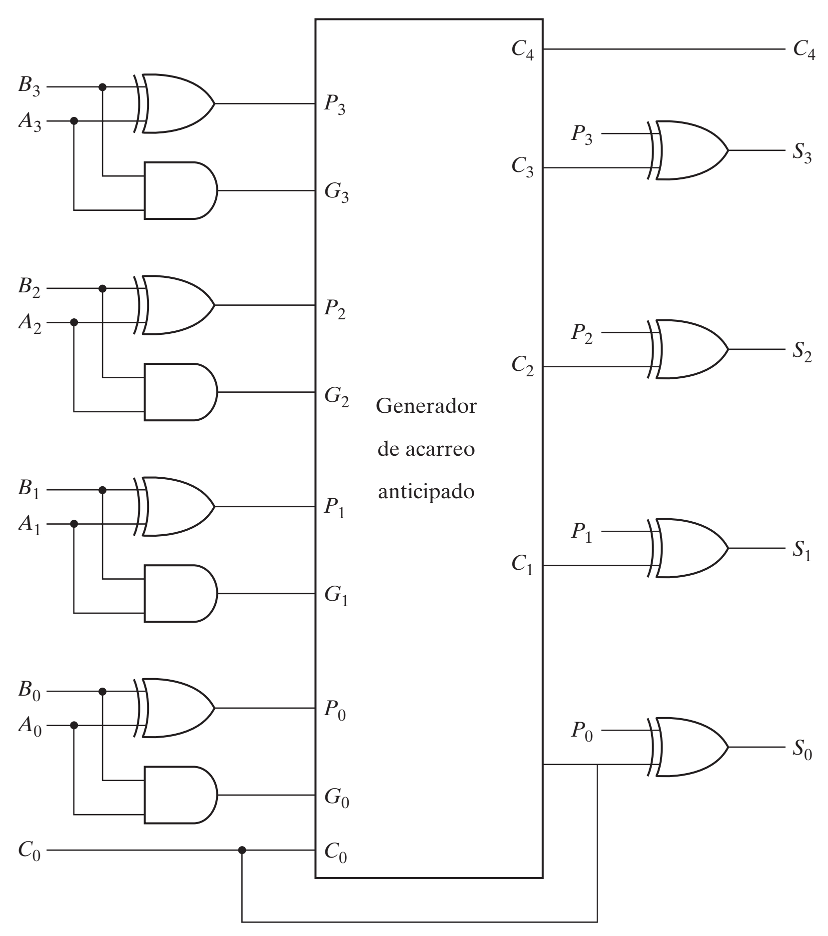


FIGURA 4-12
Sumador de cuatro bits con acarreo anticipado

Restador binario

La forma más conveniente de efectuar la resta de números binarios sin signo es utilizando complementos, como se explicó en la sección 1-5. Recuerde que la resta $A - B$ se efectúa obteniendo el complemento a dos de B y sumándolo a A. El complemento a dos se obtiene calculando el complemento a uno y sumando 1 al par de bits menos significativo. El complemento a uno se implementa con inversores, y el 1 se suma a través del acarreo de entrada.

El circuito para restar $A - B$ consiste en un sumador con inversores colocados entre cada entrada de datos B y la entrada correspondiente del sumador completo. El acarreo de entrada C_0 debe ser igual a 1 al restar. La operación se convierte entonces en A más el complemento a uno de B más 1. Esto es igual a A más el complemento a dos de B. En el caso de números sin signo, esto da $A - B$ si $A \geq B$, o el complemento a dos de $(B - A)$ si $A < B$. En el caso

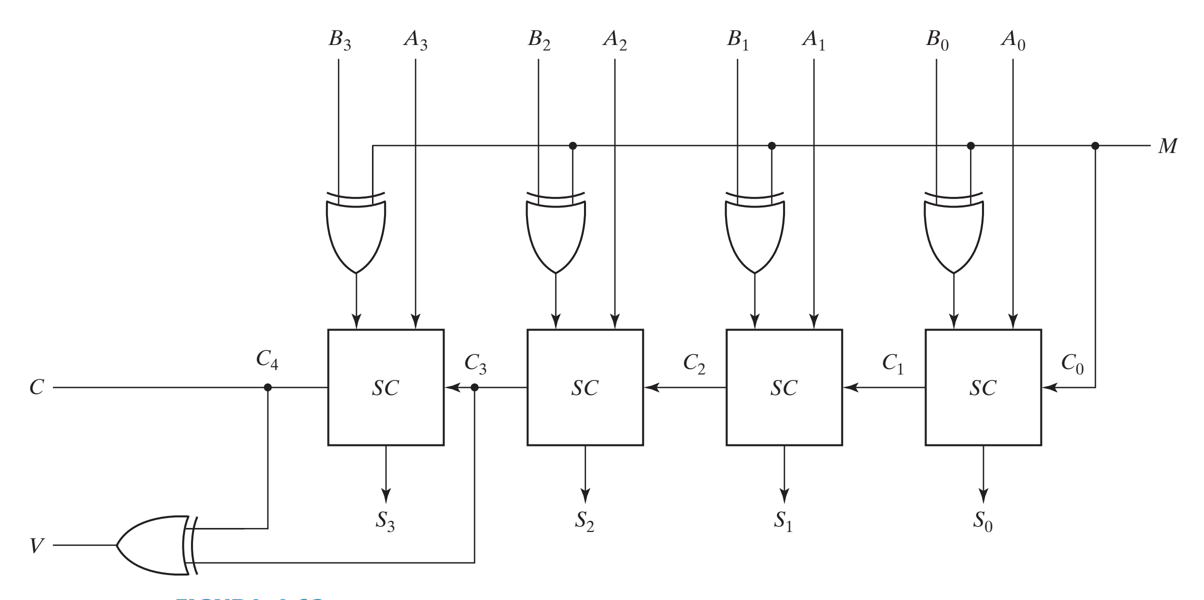


FIGURA 4-13
Sumador-restador de cuatro bits

de números con signo, el resultado es $A - B$, siempre que no haya desbordamiento. (Véase la sección 1-6.)

Las operaciones de suma y resta se pueden combinar en un solo circuito que tiene un sumador binario compartido. Esto se hace incluyendo una compuerta OR exclusivo con cada sumador completo. En la figura 4-13 se reproduce un circuito sumador-restador de cuatro bits. La entrada de modo M controla la operación. Si $M = 0$, el circuito es un sumador; y si $M = 1$, el circuito se convierte en un restador. Cada compuerta OR exclusivo recibe la entrada M y una de las entradas de B. Cuando $M = 0$, tenemos $B \oplus 0 = B$. Los sumadores completos reciben el valor de B, el acarreo de entrada es 0, y el circuito efectúa A más B. Cuando $M = 1$, tenemos $B \oplus 1 = B'$ y $C_0 = 1$. Todas las entradas de B se complementan y se suma un 1 a través del acarreo de entrada. El circuito efectúa la operación A más el complemento a dos de B. (El OR exclusivo con salida V es para detectar un desbordamiento.)

Vale la pena señalar que los números binarios en el sistema de complemento con signo se suman y restan con las mismas reglas básicas de suma y resta que los números sin signo. Por tanto, las computadoras sólo necesitan un circuito compartido en hardware para manejar ambos tipos de aritmética. El usuario o el programador deberá interpretar los resultados de tales sumas o restas de forma distinta, dependiendo de si se supone que los números tienen signo o no.

Desbordamiento

Cuando dos números de n dígitos cada uno se suman y la suma ocupa $n + 1$ dígitos, decimos que hubo un desbordamiento. Esto se cumple con los números binarios y decimales, con y sin signo. Cuando sumamos con lápiz y papel, el desbordamiento no causa problemas, porque la anchura del papel no limita la escritura de la suma. En las computadoras digitales el desbordamiento sí representa un problema porque el número de bits que contienen al número es finito, y un resultado que contiene $n + 1$ bits no cabe. Por ello, muchas computadoras detectan

cuando ocurre un desbordamiento, y "encienden" un flip-flop específico que el usuario puede verificar.

La detección de un desbordamiento después de la suma de dos números binarios depende de si se considera que los números tienen signo o no. Cuando se suman dos números sin signo, el desbordamiento se detecta en el acarreo final de la posición más significativa. En el caso de números con signo, el bit de la extrema izquierda siempre representa al signo y los números negativos están en forma de complemento a dos. Cuando se suman dos números con signo, el bit de signo se trata como parte del número y el acarreo final no indica un desbordamiento.

No existe desbordamiento después de una suma si un número es positivo y el otro es negativo, ya que la suma de un número positivo y uno negativo produce un resultado más pequeño que el mayor de los dos números originales. Podría haber un desbordamiento si los dos números sumados son ambos positivos o ambos negativos. Para entender esto, considere el ejemplo siguiente. Dos números binarios con signo, $+70$ y $+80$ se almacenan en dos registros de ocho bits. El intervalo de números al que cada registro puede dar cabida es del $+127$ binario al -128 binario. Puesto que la suma de los dos números es $+150$, excederá la capacidad de un registro de 8 bits. Esto se cumple si ambos números son positivos o negativos. A continuación mostramos las dos sumas en binario, junto con los últimos dos acarreos;

acarreos:	0 1		acarreos	1 0	
+70	0	1000110	−70	1	0111010
+80	0	1010000	−80	1	0110000
+150	1	0010110	−150	0	1101010

Observe que el resultado de ocho bits que debería haber sido positivo tiene un bit de signo negativo, y el resultado de ocho bits que debería haber sido negativo tiene un bit de signo positivo. En cambio, si tomamos el acarreo de salida de la posición de bit de signo como bit de signo del resultado, la respuesta de nueve bits así obtenida será correcta. Puesto que la respuesta no cabe en ocho bits, decimos que se ha presentado un desbordamiento.

Es posible detectar la condición de desbordamiento observando el acarreo que llega a la posición de bit de signo y el acarreo que sale de ella. Si los dos acarreos son distintos, ha habido un desbordamiento. Esto se hace evidente en los ejemplos, donde se muestran explícitamente los dos acarreos. Si estos dos acarreos se aplican a una compuerta OR exclusivo, se detectará un desbordamiento cuando la salida de esa compuerta sea 1. Para que este método funcione correctamente, es preciso calcular el complemento a dos obteniendo el complemento a uno y sumándole 1. Esto da cuenta de la condición en la que se complementa el número negativo máximo.

El circuito sumador-restador binario con salidas C y V se representa en la figura 4-13. Si se considera que los dos números binarios carecen de signo, el bit C detectará un acarreo después de la suma o un préstamo después de la resta. Si se considera que los números tienen signo, el bit V detectará un desbordamiento. Si $V = 0$ después de una suma o resta, querrá decir que no hubo desbordamiento y que el resultado de n bits es correcto. Si $V = 1$, el resultado de la operación tiene $n + 1$ bits, pero sólo los n bits de la derecha del número caben en el espacio disponible, así que ha habido un desbordamiento. El $(n + 1)$*ésimo* bit es el signo real, que ha sido desplazado de su posición.

4-5 SUMADOR DECIMAL

Las computadoras o calculadoras que realizan operaciones aritméticas directamente en el sistema numérico decimal representan los números decimales codificados en binario. Un sumador de una computadora así deberá utilizar circuitos de aritmética que acepten números decimales codificados y presenten los resultados en el mismo código. En el caso de la suma binaria, basta con considerar un par de bits significativos más un acarreo previo. Un sumador decimal requiere como mínimo nueve entradas y cinco salidas, ya que se requieren cuatro bits para codificar cada dígito decimal y el circuito necesita un acarreo de entrada y uno de salida. Hay una amplia variedad de posibles circuitos sumadores digitales, dependiendo del código empleado para representar los dígitos decimales. Aquí consideraremos un sumador decimal para el código BCD. (Véase la sección 1-7.)

Sumador BCD

Consideremos la suma aritmética de dos dígitos decimales en BCD, junto con un acarreo de entrada de una etapa anterior. Puesto que ninguno de los dígitos de entrada es mayor que 9, la suma de salida no puede ser mayor que $9 + 9 + 1 = 19$, donde el 1 de la suma es el acarreo de entrada. Suponga que aplicamos dos dígitos BCD a un sumador binario de cuatro bits. El sumador formará la suma en *binario* y producirá un resultado entre 0 y 19. Estos números binarios se presentan en forma de lista en la tabla 4-5 y se rotulan con los símbolos K, Z_8, Z_4, Z_2 y Z_1. K es el acarreo, y los subíndices de Z representan los pesos 8, 4, 2 y 1 que se pueden asignar a los cuatro bits en el código BCD. Las columnas bajo "Suma binaria" presentan el valor

Tabla 4-5
Deducción de un sumador BCD

Suma binaria					Suma BCD					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

binario que aparece en las salidas del sumador binario de cuatro bits. La suma de dos dígitos decimales se debe representar en BCD, así que deberá aparecer en la forma presentada en la columna bajo "Suma BCD". El problema consiste en encontrar una regla para convertir la suma binaria en la representación BCD correcta de la suma.

Al examinar el contenido de la tabla, será evidente que cuando la suma binaria es 1001 o menos, el número BCD correspondiente es idéntico, de modo que no es necesaria conversión alguna. Cuando la suma binaria es mayor que 1001, se obtiene una representación no válida en BCD. La suma de 6 binario (0110) a la suma binaria la convierte en la representación BCD correcta y también produce el acarreo de salida necesario.

El circuito de lógica que detecta la corrección necesaria se deduce de las entradas de la tabla. Es obvio que se necesita una corrección cuando la suma binaria tiene un acarreo de salida $K = 1$. Las otras seis combinaciones (1010 a 1111) que necesitan una corrección tienen un 1 en la posición Z_8. Para distinguirlas de los números binarios 1000 y 1001, que también tienen un 1 en la posición Z_8, especificamos además que Z_4 o Z_2 deben tener un 1. La condición para la corrección y el acarreo de salida se expresa mediante la función booleana

$$C = K + Z_8Z_4 + Z_8Z_2$$

Cuando $C = 1$, es necesario sumar 0110 a la suma binaria y generar un acarreo de salida para la etapa siguiente.

En la figura 4-14 se observa un sumador BCD que suma dos dígitos BCD y produce un dígito de suma en BCD. Primero se suman los dos dígitos decimales, junto con el acarreo de entrada, en el sumador de cuatro bits de la parte de arriba, para producir la suma binaria. Si el

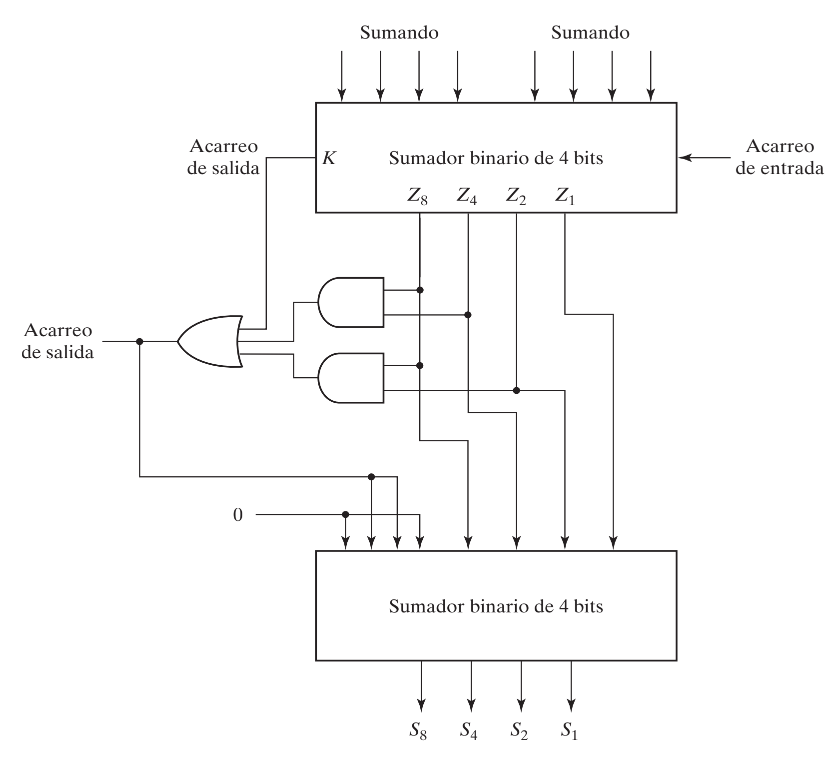


FIGURA 4-14
Diagrama de bloques de un sumador BCD

acarreo de salida es 0, no se suma nada a la suma binaria. Si es 1, se suma 0110 binario a la suma binaria con el sumador de cuatro bits de la parte de abajo. El acarreo de salida generado por el sumador de abajo se desecha, pues proporciona información con que ya se cuenta en la terminal de acarreo de salida. Un sumador decimal en paralelo que suma *n* dígitos decimales necesita *n* etapas de sumador BCD. El acarreo de salida de una etapa deberá conectarse al acarreo de entrada de la siguiente etapa de orden superior.

4-6 MULTIPLICADOR BINARIO

La multiplicación de números binarios se efectúa igual que la de números decimales. El multiplicando se multiplica por cada bit del multiplicador, comenzando por el bit menos significativo. Cada una de estas multiplicaciones forma un producto parcial. Los productos parciales sucesivos se desplazan una posición a la izquierda. El producto final se obtiene sumando los productos parciales.

Para ver cómo puede implementarse un multiplicador binario con un circuito combinacional, consideremos la multiplicación de dos números de dos bits, como se muestra en la figura 4-15. Los bits del multiplicando son B_1 y B_0, los bits del multiplicador son A_1 y A_0, y el producto es $C_3C_2C_1C_0$. El primer producto parcial se forma multiplicando A_0 por B_1B_0. La multiplicación de dos bits como A_0 y B_0 produce 1 si ambos bits son 1; de lo contrario, produce 0. Esto es idéntico a la operación AND. Por tanto, el producto parcial puede implementarse con compuertas AND como se indica en el diagrama. El segundo producto parcial se forma multiplicando A_1 por B_1B_0 y se desplaza una posición a la izquierda. Los dos productos parciales se suman con dos circuitos de semisumador (SS). Por lo regular, los productos parciales tienen más bits, y ello obliga a usar sumadores completos para obtener la suma de los productos parciales. Observe que el bit menos significativo del producto no tiene que pasar por un sumador porque se forma con la salida de la primera compuerta AND.

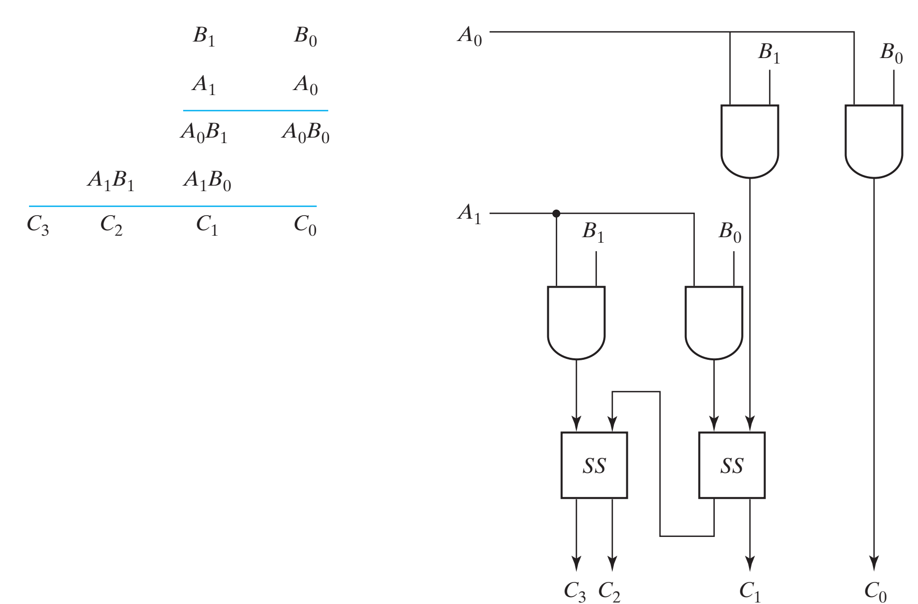


FIGURA 4-15
Multiplicador binario de dos bits por dos bits

Podemos construir de forma similar un multiplicador binario de más bits con circuitos combinacionales. Se obtiene el AND de un bit del multiplicador y cada bit del multiplicando en tantos niveles como haya bits en el multiplicador. La salida binaria de cada nivel de compuertas AND se suma al producto parcial del nivel anterior para formar un nuevo producto parcial. El último nivel genera el producto. Si el multiplicador tiene J bits y el multiplicando tiene K bits, necesitaremos $(J \times K)$ compuertas AND y $(J - 1)$ sumadores de K bits para obtener un producto de $J + K$ bits.

Como segundo ejemplo, consideremos un circuito multiplicador que multiplica un número binario de cuatro bits por uno de tres bits. Representaremos el multiplicando con $B_3B_2B_1B_0$, y el multiplicador, con $A_2A_1A_0$. Puesto que $K = 4$ y $J = 3$, necesitaremos 12 compuertas AND y dos sumadores de cuatro bits para obtener un producto de siete bits. El diagrama lógico del multiplicador se presenta en la figura 4-16.

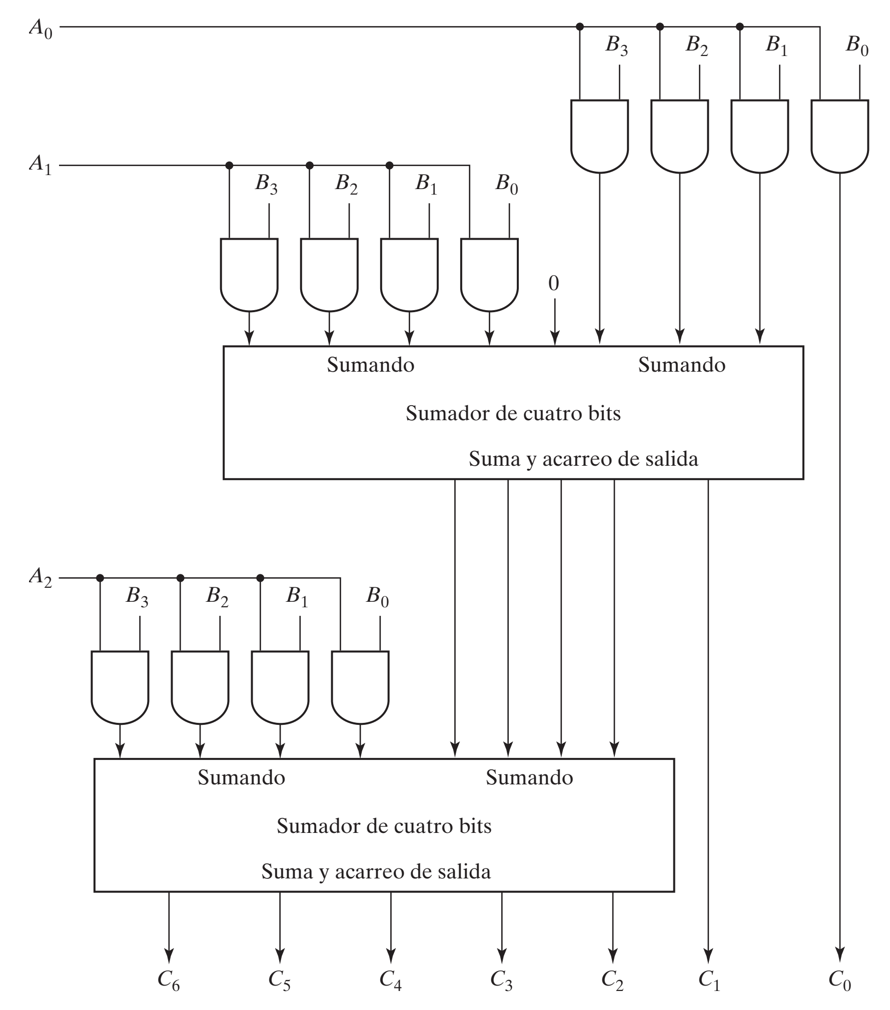


FIGURA 4-16
Multiplicador binario de 4 bits por 3 bits

4-7 COMPARADOR DE MAGNITUDES

La comparación de dos números es una operación que determina si un número es mayor que, menor que o igual a otro número. Un *comparador de magnitudes* es un circuito combinacional que compara dos números, A y B, y determina sus magnitudes relativas. El resultado de la comparación se especifica con tres variables binarias que indican si $A > B$, $A = B$ o $A < B$.

El circuito para comparar dos números de n bits tiene 2^{2n} entradas en la tabla de verdad y resulta difícil de manejar incluso con $n = 3$. Por otra parte, como el lector seguramente sospechará, los circuitos comparadores poseen cierto grado de regularidad. Las funciones digitales que poseen una regularidad inherente bien definida por lo regular se diseñan empleando un procedimiento algorítmico. Un algoritmo es un procedimiento que especifica un conjunto finito de pasos que, si se siguen, producen la solución de un problema. Ilustraremos este método deduciendo un algoritmo para el diseño de un comparador de magnitudes de cuatro bits.

El algoritmo es una aplicación directa del procedimiento que una persona sigue para comparar las magnitudes relativas de dos números. Consideremos dos números, A y B, de cuatro dígitos cada uno. Escribiremos los coeficientes de los números del más al menos significativo:

$$A = A_3A_2A_1A_0$$
$$B = B_3B_2B_1B_0$$

Cada letra con subíndice representa uno de los dígitos del número. Los dos números son iguales si todos los pares de dígitos significativos son iguales; $A_3 = B_3$ y $A_2 = B_2$ y $A_1 = B_1$ y $A_0 = B_0$. Si los números son binarios, los dígitos son 1 o 0, y la relación de igualdad de cada par de bits se expresa lógicamente con una función OR exclusivo así:

$$x_i = A_iB_i + A_i'B_i' \qquad \text{para } i = 0, 1, 2, 3$$

donde $x_i = 1$ únicamente si los dos bits de la posición i son iguales (es decir, si ambos son 1 o ambos son 0).

La igualdad de los dos números, A y B, se indica en un circuito combinacional con una variable binaria de salida que designaremos con el símbolo ($A = B$). Esta variable binaria es 1 si los números de entrada, A y B, son iguales, y es 0 en caso contrario. Para que exista la condición de igualdad, las x_i variables deberán ser todas 1. Esto implica una operación AND de todas las variables:

$$(A = B) = x_3x_2x_1x_0$$

La variable *binaria* ($A = B$) es 1 únicamente si todos los pares de dígitos de los dos números son iguales.

Para determinar si A es mayor o menor que B, se inspeccionan las magnitudes relativas de pares de dígitos significativos, comenzando por la posición más significativa. Si los dos dígitos son iguales, se comparará el siguiente par de dígitos menos significativos. Esta comparación continuará hasta encontrar un par de dígitos distintos. Si el dígito correspondiente de A es 1 y el de B es 0, concluimos que $A > B$. Si el dígito correspondiente de A es 0 y el de B es 1, sabemos que $A < B$. La comparación sucesiva se expresa lógicamente con las dos funciones booleanas

$$(A > B) = A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$$
$$(A < B) = A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1 + x_3x_2x_1A_0'B_0$$

Los símbolos ($A > B$) y ($A < B$) son variables de salida *binarias* que valen 1 cuando $A > B$ o $A < B$, respectivamente.

La implementación con compuertas de las tres variables de salida que acabamos de deducir es más sencilla de lo que parece porque implica cierta repetición. Las salidas de desigual-

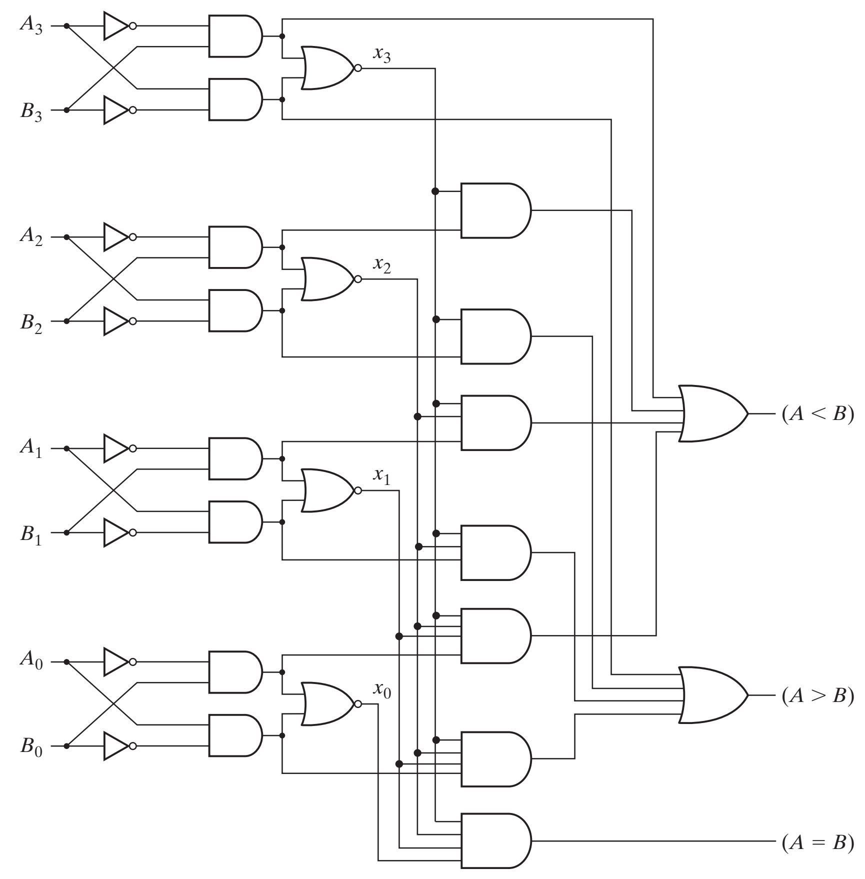


FIGURA 4-17
Comparador de magnitudes de cuatro bits

dad utilizan las mismas compuertas que generan la salida de igualdad. El diagrama lógico del comparador de magnitudes de cuatro bits se reproduce en la figura 4-17. Las cuatro salidas x se generan con circuitos NOR exclusivo y se aplican a una compuerta AND para dar la variable binaria de salida $(A = B)$. Las otras dos salidas utilizan las variables x para generar las funciones booleanas que presentamos antes. Ésta es una implementación multinivel y sigue un patrón regular. El procedimiento para obtener circuitos comparadores de magnitud para números binarios de más de cuatro bits se deduce fácilmente de este ejemplo.

4-8 DECODIFICADORES

En los sistemas digitales, las cantidades discretas de información se representan con códigos binarios. Un código binario de n bits puede representar hasta 2^n elementos distintos de información codificada. Un *decodificador* es un circuito combinacional que convierte información binaria de n líneas de entrada a un máximo de 2^n líneas de salida distintas. Si la información codificada en n bits tiene combinaciones que no se usan, el decodificador podría tener menos de 2^n salidas.

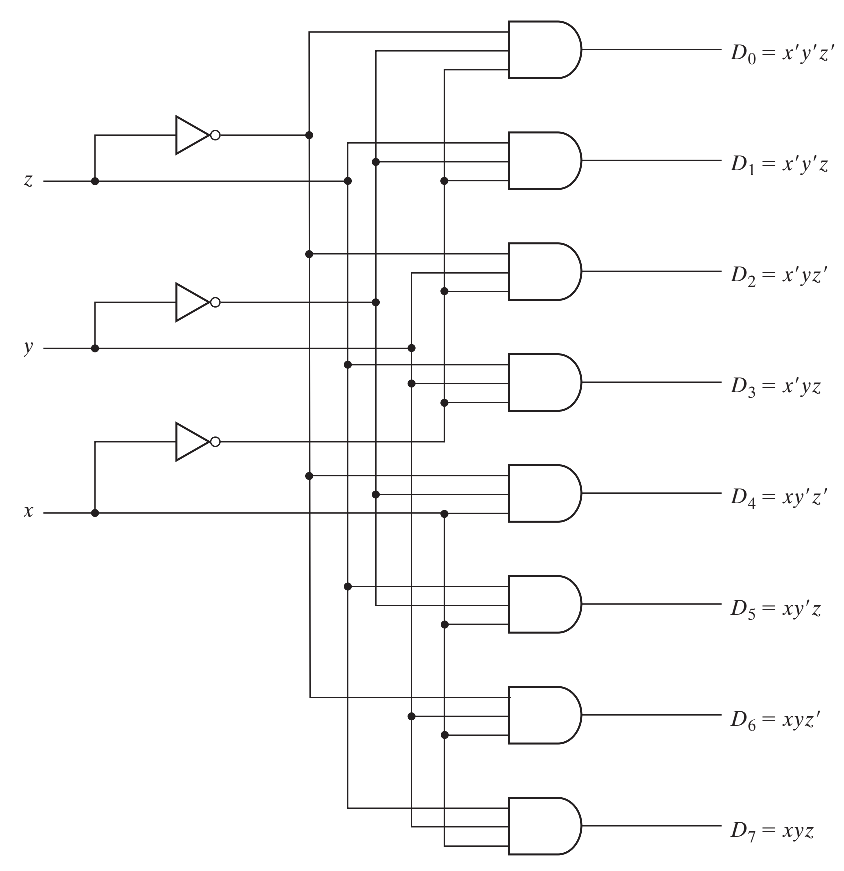


FIGURA 4-18
Decodificador de 3 a 8 líneas

Los decodificadores que presentamos aquí se describen como decodificadores de n a m líneas, donde $m \leq 2^n$. Su propósito es generar los 2^n (o menos) minitérminos de n variables de entrada. El nombre *decodificador* también se usa para referirse a otros convertidores de códigos, como un decodificador de BCD a siete segmentos.

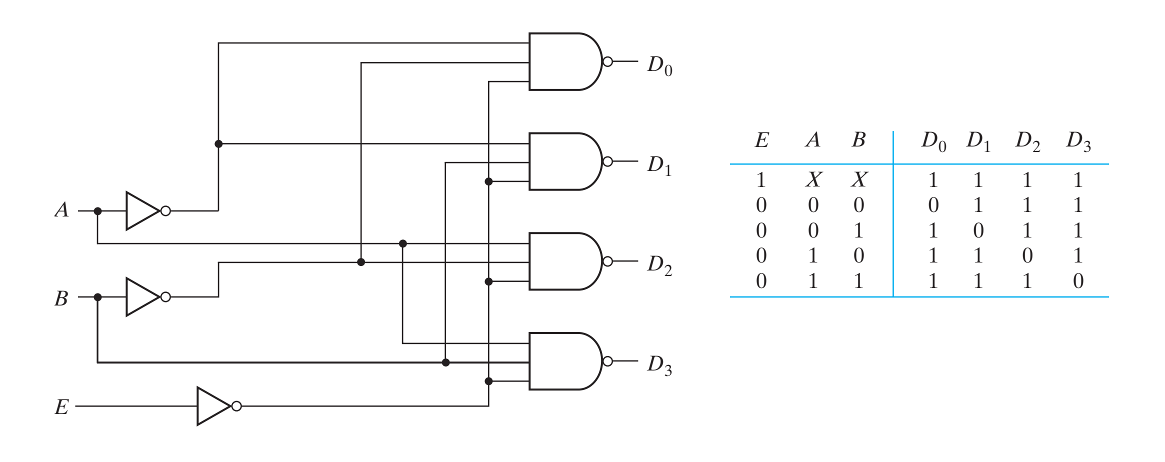
Como ejemplo, consideremos el circuito decodificador de 3 a 8 líneas de la figura 4-18. Las tres entradas se decodifican para dar ocho salidas, cada una de las cuales representa uno de los minitérminos de las tres variables de entrada. Los tres inversores producen el complemento de las entradas, y cada una de las ocho compuertas AND genera uno de los minitérminos. Una aplicación específica de este decodificador es la conversión de binario a octal. Las variables de entrada representan un número binario, y las salidas, los ocho dígitos del sistema numérico octal. Sin embargo, un decodificador de 3 a 8 líneas puede servir para decodificar cualquier código de tres bits y obtener ocho salidas, una por cada elemento del código.

El funcionamiento del decodificador podría aclararse al examinar la tabla de verdad de la tabla 4-6. Para cada posible combinación de entrada, hay siete salidas que son 0 y sólo una igual a 1. La salida que vale 1 representa el minitérmino equivalente al número binario que se está alimentando a las líneas de entrada.

Tabla 4-6
Tabla de verdad de un decodificador de 3 a 8 líneas

Entradas			Salidas							
x	*y*	*z*	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Algunos decodificadores se construyen con compuertas NAND. Puesto que una compuerta NAND produce la operación AND con la salida invertida, resulta más económico generar los minitérminos del decodificador en su forma complementada. Además, los decodificadores incluyen una o más entradas *habilitadoras* (*enable*) que controlan el funcionamiento del circuito. En la figura 4-19 se aprecia un decodificador de 2 a 4 líneas con entrada de habilitación, construido con compuertas NAND. El circuito opera con salidas complementadas y una entrada de habilitación complementada. El decodificador se habilita cuando $E = 0$. Como indica la tabla de verdad, sólo una salida puede ser 0 en cualquier momento dado; todas las demás salidas son 1. La salida cuyo valor es 0 representa el minitérmino seleccionado por las entradas A y B. El circuito queda inhabilitado cuando $E = 1$, sean cuales sean los valores de las otras dos entradas. Cuando el circuito está inhabilitado, ninguna de las salidas es 0 y ninguno de los minitér-



E	*A*	*B*	D_0	D_1	D_2	D_3
1	*X*	*X*	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

a) Diagrama lógico b) Tabla de verdad

FIGURA 4-19
Decodificador de 2 a 4 líneas con entrada habilitadora

minos está seleccionado. En general, un decodificador podría operar con salidas complementadas o no complementadas. La entrada de habilitación podría activarse con una señal 0 o con una señal 1. Algunos decodificadores tienen dos o más entradas de habilitación que deben satisfacer una condición lógica dada para habilitar el circuito.

Un decodificador con entrada de habilitación puede funcionar como desmultiplexor. Un *desmultiplexor* es un circuito que recibe información de una sola línea y la dirige a una de 2^n posibles líneas de salida. La selección de una salida específica se controla con la combinación de bits de n líneas de selección. El decodificador de la figura 4-19 funciona como desmultiplexor de 1 a 4 líneas si E se toma como una línea de entrada de datos, y A y B se toman como entradas de selección. La variable única de entrada E tiene un camino a las cuatro salidas, pero la información de entrada se dirige a sólo una de las líneas de salida, especificada por la combinación binaria de las dos líneas de selección A y B. Esto se verifica examinando la tabla de verdad del circuito. Por ejemplo, si las líneas de selección $AB = 10$, la salida D_2 tendrá el mismo valor que la entrada E, mientras que todas las demás salidas se mantendrán en 1. Dado que se obtienen operaciones de decodificador y desmultiplexor con el mismo circuito, decimos que un decodificador con entrada de habilitación es un *decodificador/desmultiplexor*.

Es posible conectar los decodificadores con entradas de habilitación unos con otros para formar un circuito decodificador más grande. La figura 4-20 muestra dos decodificadores de 3 a 8 líneas con entradas de habilitación conectadas para formar un decodificador de 4 a 16 líneas. Cuando $w = 0$, el decodificador de arriba está habilitado y el otro está inhabilitado. Todas las salidas del decodificador de abajo son 0, y las ocho salidas del generador de arriba generan los minitérminos 0000 a 0111. Cuando $w = 1$, las condiciones de habilitación se invierten; las salidas del decodificador de abajo generan los minitérminos 1000 a 1111, mientras que todas las salidas del decodificador de arriba son 0. Este ejemplo ilustra la utilidad de las entradas de habilitación en los decodificadores y otros componentes de lógica combinacional. En general, las entradas de habilitación son una característica conveniente para interconectar dos o más componentes estándar y así expandir el componente a una función similar con más entradas y salidas.

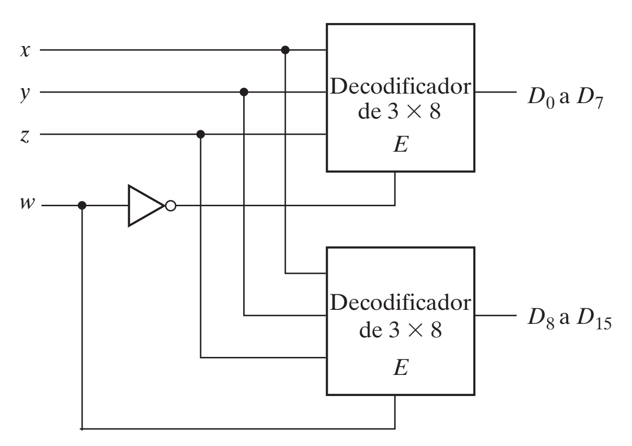


FIGURA 4-20
Decodificador 4 × 16 construido con dos decodificadores 3 × 8

Implementación de lógica combinacional

Un decodificador produce los 2^n minitérminos de n variables de entrada. Puesto que cualquier función booleana es susceptible de expresarse como suma de minitérminos, es posible utilizar un decodificador para generar los minitérminos y una compuerta OR externa para formar la suma lógica. Así, cualquier circuito combinacional con n entradas y m salidas se puede implementar con un decodificador de n a 2^n líneas y m compuertas OR.

El procedimiento para implementar un circuito combinacional con un decodificador y compuertas OR requiere expresar la función booleana del circuito como suma de minitérminos. Entonces se escoge un decodificador que genere todos los minitérminos de las variables de entrada. Las entradas a cada compuerta OR se escogen de entre las salidas del decodificador, de acuerdo con la lista de minitérminos de cada función. Ilustraremos este procedimiento con un ejemplo que implementa un circuito sumador completo.

De la tabla de verdad del sumador completo (véase la tabla 4-4), obtenemos las funciones para el circuito combinacional en forma de suma de minitérminos:

$$S(x, y, z) = \sum(1, 2, 4, 7)$$
$$C(x, y, z) = \sum(3, 5, 6, 7)$$

Puesto que hay tres entradas y un total de ocho minitérminos, se necesita un decodificador de 3 a 8 líneas. La implementación se muestra en la figura 4-21. El decodificador genera los ocho minitérminos para x, y, z. La compuerta OR de la salida S forma la suma lógica de los minitérminos 1, 2, 4 y 7. La compuerta OR de la salida C forma la suma lógica de los minitérminos 3, 5, 6 y 7.

Una función con una lista larga de minitérminos requerirá una compuerta OR con un gran número de entradas. Una función con una lista de k minitérminos se expresa en su forma complementada F' empleando $2^n - k$ minitérminos. Si el número de minitérminos de una función es mayor que $2^n/2$, podremos expresar F' con menos minitérminos. En tal caso, resulta ventajoso utilizar una compuerta NOR para sumar los minitérminos de F'. La salida de la compuerta NOR complementa esta suma y genera la salida normal F. Si se usan compuertas NAND para el decodificador, como en la figura 4-19, las compuertas externas deberán ser NAND en lugar de OR. El motivo es que un circuito de compuertas NAND de dos niveles implementa una función de suma de minitérminos y equivale a un circuito AND-OR de dos niveles.

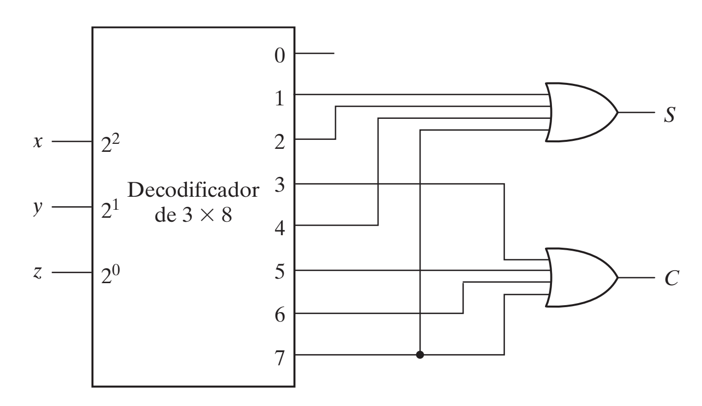


FIGURA 4-21
Implementación de un sumador completo con un decodificador

4-9 CODIFICADORES

Un codificador es un circuito digital que efectúa la operación inversa de la que efectúa un decodificador. El codificador tiene 2^n (o menos) líneas de entrada y n líneas de salida. Estas últimas generan el código binario correspondiente al valor de entrada. Un ejemplo de codificador es el codificador de octal a binario cuya tabla de verdad se presenta en la tabla 4-7. Tiene ocho entradas (una para cada uno de los dígitos octales) y tres salidas que generan el número binario correspondiente. Se supone que sólo una entrada es igual a 1 en cualquier momento dado.

El codificador se puede implementar con compuertas OR cuyas salidas se determinan directamente de la tabla de verdad. La salida z es 1 cuando el dígito octal de entrada es 1, 3, 5 o 7. La salida y es 1 para los dígitos octales 2, 3, 6 o 7, y la salida x es 1 para los dígitos 4, 5, 6 o 7. Estas condiciones se expresan con las funciones booleanas de salida siguientes:

$$z = D_1 + D_3 + D_5 + D_7$$
$$y = D_2 + D_3 + D_6 + D_7$$
$$x = D_4 + D_5 + D_6 + D_7$$

El codificador se implementa con tres compuertas OR.

El codificador definido en la tabla 4-7 tiene la limitación de que sólo una entrada puede estar activa en un momento dado. Si dos entradas están activas simultáneamente, la salida producirá una combinación no definida. Por ejemplo, si D_3 y D_6 son 1 simultáneamente, la salida del decodificador será 111 porque las tres salidas son 1. Esto no representa ni el 3 binario ni el 6 binario. Para resolver esta ambigüedad, los circuitos codificadores deben establecer una prioridad de entrada que garantice que sólo se codificará una de las entradas. Si establecemos que las entradas con subíndice más alto tienen prioridad, y si tanto D_3 como D_6 son 1 al mismo tiempo, la salida será 110 porque D_6 tiene prioridad sobre D_3.

Otra ambigüedad en el codificador de octal a binario es que se genera una salida de tres ceros cuando todas las entradas son 0; esta salida es la misma que se produce cuando D_0 es igual a 1. La discrepancia se resuelve añadiendo una salida más que indique que por lo menos una entrada es 1.

Tabla 4-7
Tabla de verdad del codificador de octal a binario

Entradas								**Salidas**		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Tabla 4-8
Tabla de verdad de un codificador con prioridad

Entradas				Salidas		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Codificador con prioridad

Un codificador con prioridad es un circuito codificador que incluye la función de prioridad. Su funcionamiento es tal que, si dos o más entradas son 1 al mismo tiempo, la salida prioritaria tendrá precedencia. En la tabla 4-8 se presenta la tabla de verdad de un codificador de cuatro entradas con prioridad. Además de las dos salidas, x y y, el circuito tiene una tercera salida designada V; ésta es un indicador de bit *válido* que adquiere el valor 1 cuando una o más entradas son 1. Si todas las entradas son 0, la entrada no será válida y V será 0. En tal caso, las otras dos salidas no se inspeccionarán y se especifican como condiciones de indiferencia. Advierta que, aunque las **X** en las columnas de salida representan condiciones de indiferencia, las **X** de las columnas de entrada sirven para representar una tabla de verdad en forma condensada. En vez de presentar los 16 minitérminos de cuatro variables, la tabla de verdad usa una **X** para representar tanto 1 como 0. Por ejemplo, X100 representa los dos minitérminos 0100 y 1100.

Según la tabla 4-8, cuanto más alto sea el subíndice de una entrada, mayor prioridad tendrá esa entrada. La entrada D_3 es la de mayor prioridad, así que, si es 1, la salida xy será 11 (3 binario) sin importar qué valor tengan las demás entradas. D_2 está en el siguiente nivel de prioridad. La salida es 10 si $D_2 = 1$, siempre que $D_3 = 0$ e independientemente del valor que tengan las otras dos entradas de menor prioridad. La salida para D_1 sólo se genera si las entradas con mayor prioridad son 0, y así hasta el nivel de prioridad más bajo.

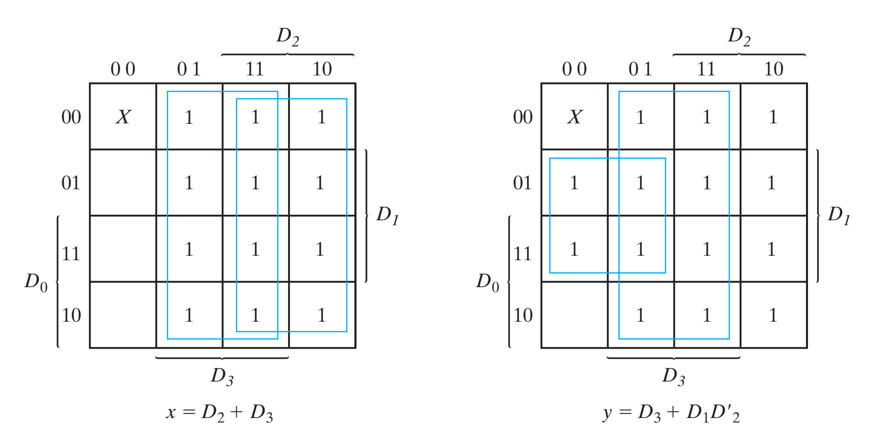


FIGURA 4-22
Mapas para el codificador con prioridad

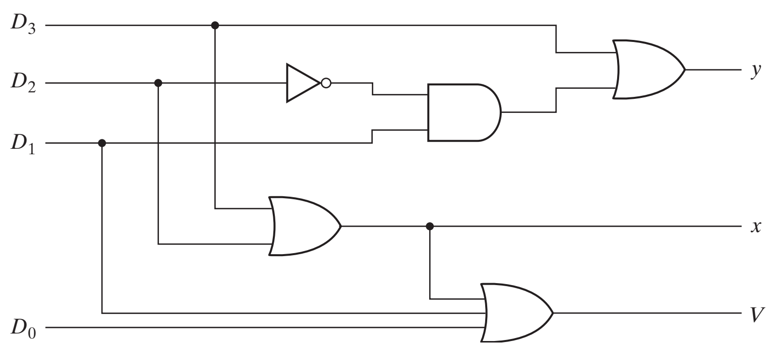


FIGURA 4-23
Codificador de cuatro entradas con prioridad

Los mapas para simplificar las salidas x y y aparecen en la figura 4-22. Los minitérminos para las dos funciones se deducen a partir de la tabla 4-8. Aunque la tabla sólo tiene cinco filas, al sustituir cada X de una fila, primero por 0 y después por 1, obtendremos las 16 posibles combinaciones de entrada. Por ejemplo, la cuarta fila de la tabla, que tiene XX10, representa los cuatro minitérminos 0010, 0110, 1010 y 1110. Las expresiones booleanas simplificadas para el codificador con prioridad se obtienen de los mapas. La condición para la salida V es una función OR de todas las variables de entrada. El codificador con prioridad se implementa en la figura 4-23 de acuerdo con las funciones booleanas siguientes:

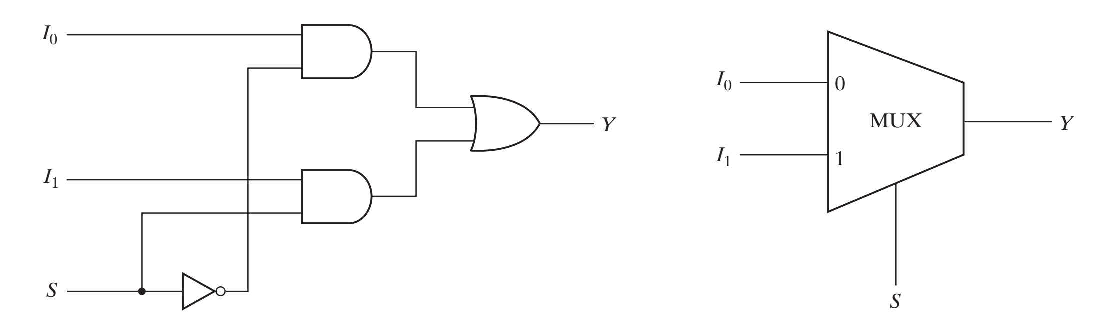
$$x = D_2 + D_3$$
$$y = D_3 + D_1 D_2'$$
$$V = D_0 + D_1 + D_2 + D_3$$

4-10 MULTIPLEXORES

Un multiplexor es un circuito combinacional que selecciona información binaria de una de muchas líneas de entrada y la envía a una sola línea de salida. La selección de una línea de entrada dada se controla con un conjunto de líneas de selección. Normalmente, hay 2^n líneas de entrada y n líneas de selección cuyas combinaciones de bits determinan cuál entrada se selecciona.

Un multiplexor de 2 líneas a 1 conecta una de dos fuentes de un bit a un destino común, como se indica en la figura 4-24. El circuito tiene dos líneas de entrada de datos, una línea de salida y una línea de selección S. Cuando $S = 0$, se habilita la compuerta AND de arriba e I_0 cuenta con una trayectoria hacia la salida. Cuando $S = 1$, la compuerta AND inferior está habilitada e I_1 tiene una trayectoria hacia la salida. El multiplexor actúa como un interruptor electrónico que selecciona una de dos fuentes. El diagrama de bloques de un multiplexor a veces se representa con un símbolo en forma de cuña, como en la figura 4-24b). Esto sugiere visualmente cómo una fuente de datos, seleccionada de entre varias, se dirige a un solo destino. En los diagramas de bloques es común rotular los multiplexores como MUX.

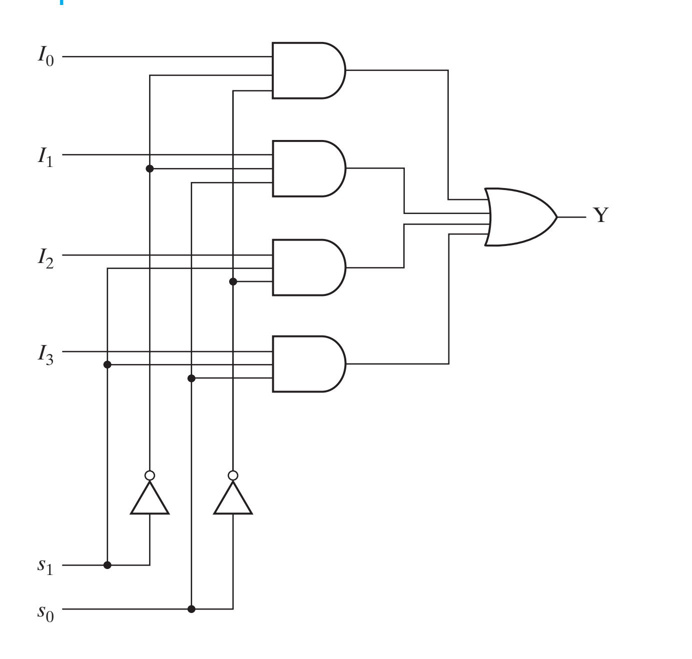
En la figura 4-25 se presenta un multiplexor de 4 líneas a 1. Cada una de las cuatro entradas, I_0 a I_3, se aplica a una entrada de una compuerta AND. Las líneas de selección S_1 y S_0 se decodifican para seleccionar una compuerta AND determinada. Las salidas de las compuertas AND se aplican a una sola compuerta OR que genera la salida de una sola línea. La tabla de la función indica qué entrada se pasa a la salida con cada combinación de los valores binarios de selección. Para ilustrar el funcionamiento del circuito, consideremos el caso en que



a) Diagrama lógico

b) Diagrama de bloque

FIGURA 4-24
Multiplexor de 2 líneas a 1



s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

a) Diagrama lógico

b) Tabla de función

FIGURA 4-25
Multiplexor de 4 líneas a 1

$S_1 S_0 = 10$. La compuerta AND asociada a la entrada I_2 tiene 1 en dos de sus entradas, y la tercera conectada a I_2. Las otras tres compuertas AND tienen 0 en por lo menos una de sus entradas, lo que hace que produzcan 0 como salida. Así, la salida de la compuerta OR tiene el mismo valor que I_2, así que constituye un camino de la entrada seleccionada hasta la salida. Los multiplexores también se denominan *selectores de datos*, pues seleccionan una de varias entradas y dirigen la información binaria a la línea de salida.

Las compuertas AND y los inversores del multiplexor semejan un circuito decodificador y, de hecho, decodifican las líneas de selección de entrada. En general, un multiplexor de 2^n líneas

a 1 se construye a partir de un decodificador de n a 2^n líneas añadiéndole 2^n líneas de entrada, una para cada compuerta AND. Las salidas de las compuertas AND se aplican a una sola compuerta OR. El tamaño del multiplexor se especifica con el número de líneas de entrada de datos que tiene (2^n) y la única línea de salida. El número de líneas de selección (n) está implícito en el número de líneas de datos (2^n). Al igual que los decodificadores, los multiplexores podrían tener una entrada de habilitación que controla el funcionamiento de la unidad. Si dicha entrada está inactiva, las salidas quedarán inhabilitadas, y si está activa, el circuito funcionará como un multiplexor normal.

Los circuitos multiplexores se pueden combinar con entradas de selección comunes para crear una lógica de selección de múltiples bits. Por ejemplo, en la figura 4-26 se ilustra un multiplexor cuádruple de 2 líneas a 1. El circuito tiene cuatro multiplexores, cada uno de los

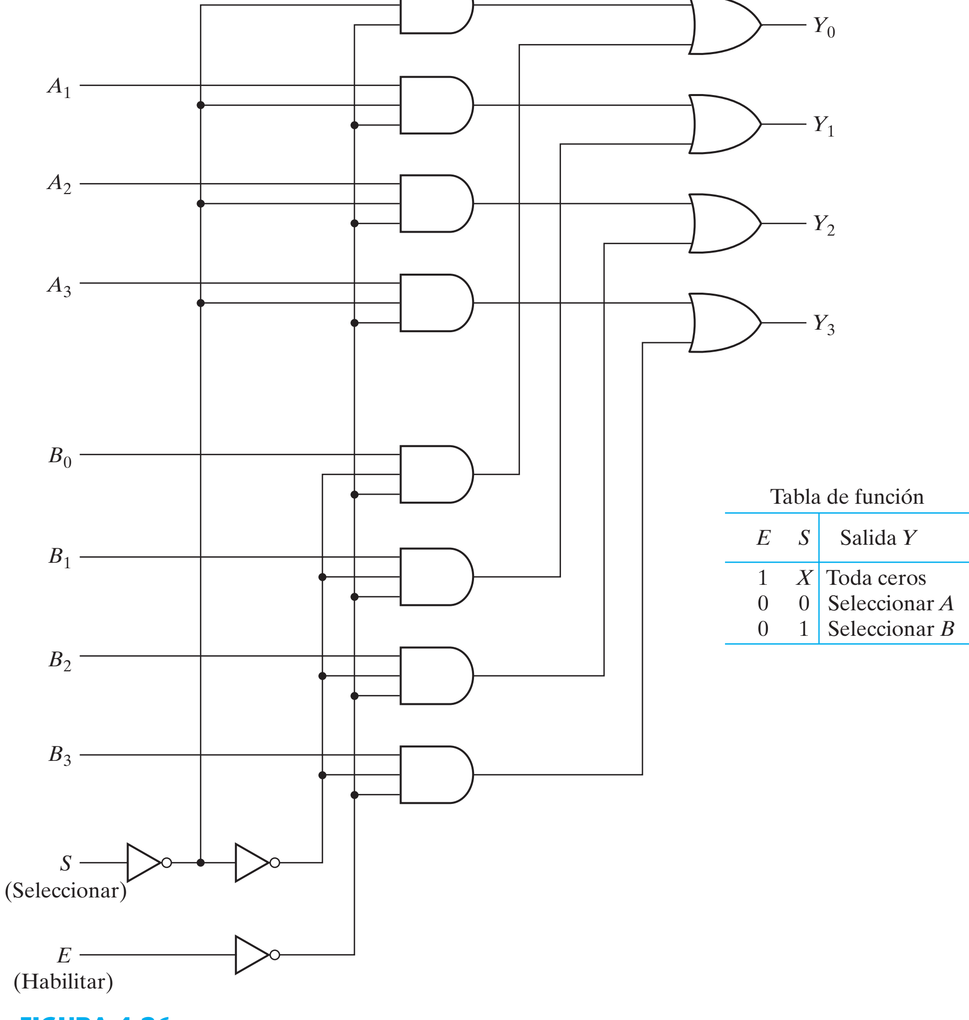


Tabla de función

E	S	Salida Y
1	X	Toda ceros
0	0	Seleccionar A
0	1	Seleccionar B

FIGURA 4-26
Multiplexor cuádruple de 2 líneas a 1

cuales puede seleccionar una de dos líneas de entrada. Podemos escoger que la salida Y_0 provenga de la entrada A_0 o bien de B_0. De igual manera, la salida Y_1 podría tener el valor de A_1 o B_1, y así sucesivamente. La línea de selección de entrada S selecciona una de las líneas en cada uno de los cuatro multiplexores. La entrada de habilitación E debe estar activa para que el funcionamiento sea normal. Aunque el circuito contiene cuatro multiplexores de 2 líneas a 1, seguramente lo veremos como un circuito que selecciona uno de dos conjuntos de líneas de datos de cuatro bits. Como indica la tabla de función, la unidad se habilita cuando $E = 0$. Entonces, si $S = 0$, las cuatro entradas A tendrán un camino a las cuatro salidas. En cambio, si $S = 1$, las cuatro entradas B se aplicarán a las salidas. Cuando $E = 1$, todas las salidas tienen 0, sin importar qué valor tenga S.

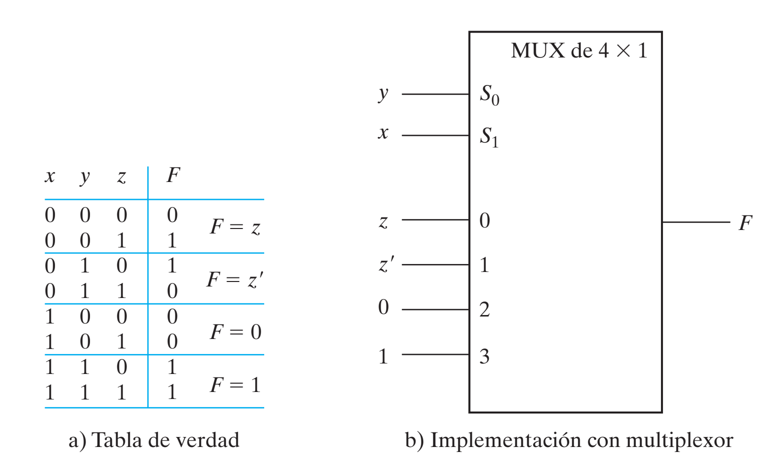
Implementación de funciones booleanas

En la sección 4-8 se explicó cómo utilizar un decodificador para implementar funciones booleanas añadiendo compuertas OR externas. Un examen del diagrama lógico de un multiplexor revela que básicamente es un decodificador con una compuerta OR incluida en la unidad. Los minitérminos de una función se generan en un multiplexor mediante el circuito asociado a las entradas de selección. Los minitérminos individuales se pueden seleccionar con las entradas de datos. Esto ofrece un método para implementar una función booleana de n variables con un multiplexor que tiene n entradas de selección y 2^n entradas de datos, una para cada minitérmino.

Ahora mostraremos un método más eficiente para implementar una función booleana de n variables con un multiplexor que tiene $n - 1$ entradas de selección. Las primeras $n - 1$ variables de la función se conectan a las entradas de selección del multiplexor. La variable restante de la función se utiliza para las entradas de datos. Si denotamos esa variable con z, cada entrada de datos del multiplexor será, z, z', 1 o 0. Para ilustrar este procedimiento, consideremos la función booleana de tres variables:

$$F(x, y, z) = \sum(1, 2, 6, 7)$$

La función puede implementarse con un multiplexor de 4 líneas a 1 como se indica en la figura 4-27. Las dos variables x y y se aplican a las líneas de selección en ese orden; x se conecta



x	y	z	F	
0	0	0	0	F = z
0	0	1	1	
0	1	0	1	F = z′
0	1	1	0	
1	0	0	0	F = 0
1	0	1	0	
1	1	0	1	F = 1
1	1	1	1	

a) Tabla de verdad

b) Implementación con multiplexor

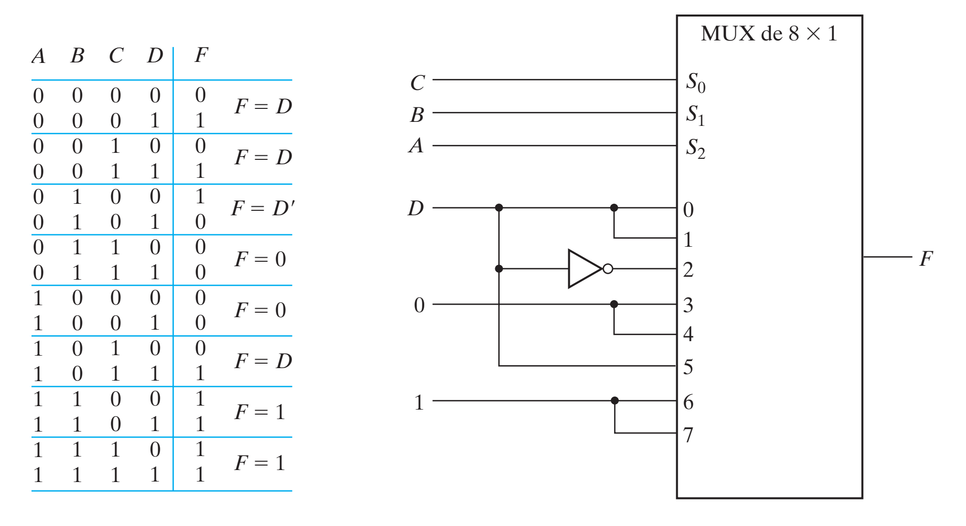
FIGURA 4-27
Implementación de una función booleana con un multiplexor

a la entrada S_1 y y se conecta a S_0. Los valores de las líneas de entrada de datos se deducen de la tabla de verdad de la función. Cuando $xy = 00$, la salida F es igual a z porque $F = 0$ cuando $z = 0$ y $F = 1$ cuando $z = 1$. Esto requiere aplicar la variable z a la entrada de datos 0. El funcionamiento del multiplexor es tal que, cuando $xy = 00$, la entrada de datos 0 tiene una trayectoria hacia la salida y eso hace que F sea igual a z. De forma similar, podemos determinar las entradas que deben recibir las líneas de datos 1, 2 y 3, a partir del valor de F cuando $xy = 01$, 10 y 11, respectivamente. Este ejemplo específico muestra las cuatro posibilidades que podemos tener en las entradas de datos.

El procedimiento general para implementar cualquier función booleana de n variables con un multiplexor de $n - 1$ entradas de selección y 2^{n-1} entradas de datos se deduce del ejemplo anterior. Primero se enumera la función booleana en una tabla de verdad. Las primeras $n - 1$ variables de la tabla se aplican a las entradas de selección del multiplexor. Para cada combinación de las variables de selección, evaluamos la salida en función de la última variable. Esta función puede ser 0, 1, la variable o el complemento de la variable. Luego, estos valores se aplican a las entradas de datos en el orden correcto. Como segundo ejemplo, consideremos la implementación de la función booleana

$$F(A, B, C, D) = \sum(1, 3, 4, 11, 12, 13, 14, 15).$$

Esta función se implementa con un multiplexor con tres entradas de selección, como se ilustra en la figura 4-28. Observe que la primera variable, A, debe conectarse a la entrada de selección S_2 para que A, B y C correspondan a las entradas de selección S_2, S_1 y S_0, respectivamente. Los valores de las entradas de datos se determinan de la tabla de verdad que se presenta en la figura. El número de línea de datos correspondiente se determina a partir de la combinación binaria de ABC. Por ejemplo, cuando $ABC = 101$, la tabla indica que $F = D$, así que se aplica la variable de entrada D a la entrada de datos 5. Las constantes binarias 0 y 1 corresponden a dos valores de señal fijos. Cuando se usan circuitos integrados, el 0 lógico corresponde a la tierra de señal y el 1 lógico equivale a la señal de potencia, que por lo regular es de 5 volts.



A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	

FIGURA 4-28
Implementación de una función de cuatro entradas con un multiplexor

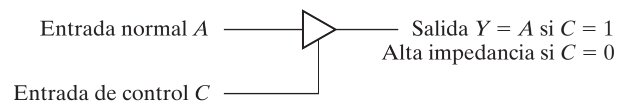


FIGURA 4-29
Símbolo gráfico para un búfer de tres estados

Compuertas de tres estados

Es posible construir un multiplexor con compuertas de tres estados. Una compuerta de tres estados es un circuito digital que exhibe tres estados. Dos de los estados son señales equivalentes al 1 y al 0 lógicos, como en las compuertas convencionales. El tercer estado es un estado de alta impedancia. El *estado de alta impedancia* se comporta como un circuito abierto, lo que implica que la salida parece estar desconectada y el circuito carece de significado lógico. Las compuertas de tres estados son capaces de realizar cualquier lógica convencional, como AND o NAND, pero la que se usa más comúnmente es la compuerta búfer.

En la figura 4-29 se observa el símbolo gráfico de un búfer de tres estados. Se distingue de un búfer normal con una línea de control de entrada que entra por la parte inferior del símbolo de compuerta. El búfer tiene una entrada normal, una salida y una entrada de control que determina el estado de la salida. Si la entrada de control es 1, la salida está habilitada y la compuerta se comporta como un búfer convencional, cuya salida es igual a la entrada normal. Cuando la entrada de control es 0, la salida se inhabilita y la compuerta pasa a un estado de alta impedancia, sea cual sea el valor en la entrada normal. El estado de alta impedancia de una compuerta de tres estados ofrece una característica especial que no ofrecen otras compuertas. Gracias a ella, un gran número de salidas de compuertas de tres estados se pueden conectar con alambres para formar una línea común sin correr riesgos por los efectos de carga.

En la figura 4-30 se ilustra la construcción de multiplexores con búferes de tres estados. La parte a) de la figura muestra la construcción de un multiplexor de 2 líneas a 1 con dos búferes

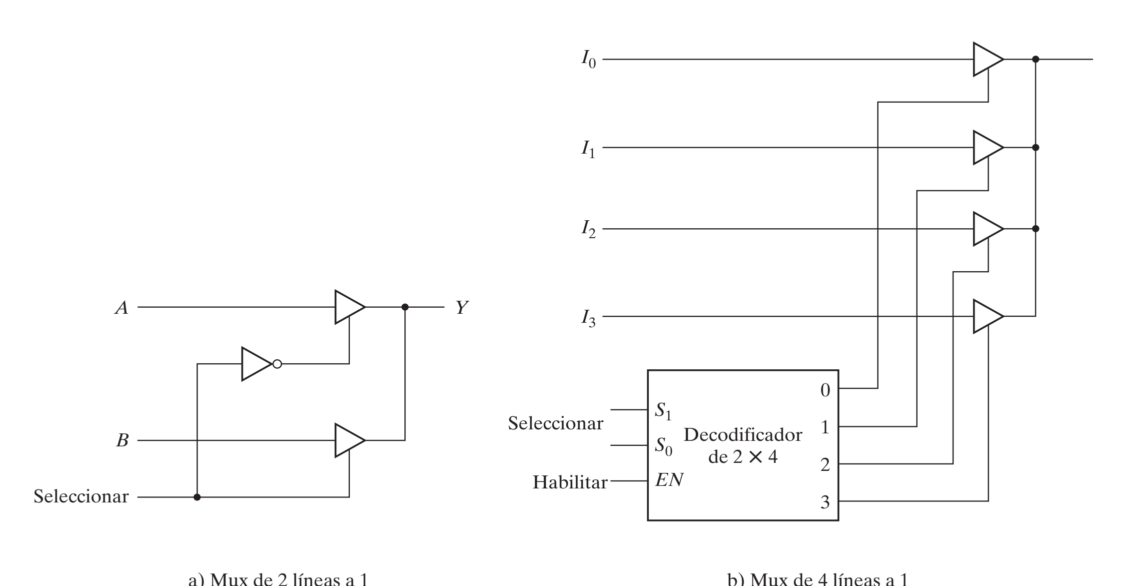


FIGURA 4-30
Multiplexores con compuertas de tres estados

de tres estados y un inversor. Las dos salidas se conectan entre sí para formar una sola línea de salida. (Cabe señalar que este tipo de conexión no puede efectuarse con compuertas que no tengan salidas de tres estados.) Si la entrada de selección es 0, el búfer superior queda habilitado por su entrada de control, y el búfer inferior queda inhabilitado. Entonces, la salida Y es igual a la entrada A. Cuando la entrada de selección es 1, se habilita el búfer inferior y Y es igual a B.

En la figura 4-30b) se muestra la construcción de un multiplexor de 4 líneas a 1. Las salidas de cuatro búferes de tres estados se conectan entre sí para formar una sola línea de salida. Las entradas de control de los búferes determinan cuál de las cuatro entradas normales, I_0 a I_3, se conectará a la línea de salida. Nunca habrá más de un búfer en el estado activo a la vez. Los búferes conectados deben controlarse de modo que sólo un búfer de tres estados tenga acceso a la salida, mientras todos los demás búferes se mantienen en un estado de alta impedancia. Una forma de garantizar que no más de una entrada de control esté activa en un momento dado es utilizar un decodificador, como se indica en el diagrama. Si la entrada de habilitación del decodificador es 0, sus cuatro salidas son 0, y la línea de bus está en un estado de alta impedancia porque los cuatro búferes están inhabilitados. Cuando la entrada de habilitación está activa, uno de los búferes de tres estados estará activo, dependiendo del valor binario en las entradas de selección del decodificador. Una investigación cuidadosa revelará que este circuito es otra forma de construir un multiplexor de 4 líneas a 1.

4-11 HDL PARA CIRCUITOS COMBINACIONALES

Presentamos el lenguaje de descripción de hardware (HDL) Verilog en la sección 3-9. En esta sección se hablará de las opciones existentes para describir circuitos combinacionales en HDL. Presentaremos los circuitos secuenciales en el capítulo siguiente. Como ya se dijo, el módulo es el bloque de construcción básico en Verilog HDL. Es posible describir un módulo con cualquiera de las siguientes técnicas de modelado (o mediante una combinación de ellas):

- Modelado en el nivel de compuertas creando ejemplares de compuertas primitivas y módulos definidos por el usuario.
- Modelado de flujo de datos empleando enunciados de asignación continua con la palabra clave **assign.**
- Modelado de comportamiento utilizando enunciados de asignación procedimentales con la palabra clave **always.**

El modelado en el nivel de compuertas describe el circuito especificando las compuertas y cómo se conectan entre sí. El modelado de flujo de datos se utiliza primordialmente para describir circuitos combinacionales. El modelado de comportamiento sirve para describir sistemas digitales en un nivel de abstracción más alto. Existe otro estilo de modelado llamado modelado en el nivel de interruptores, que permite diseñar en el nivel de transistores MOS y que se analizará en la sección 10-10.

Modelado en el nivel de compuertas

Ya presentamos el modelado en el nivel de compuertas con un ejemplo sencillo en la sección 3-9. En este tipo de representación, los circuitos se especifican por sus compuertas lógicas y su interconexión. Es una descripción textual de un diagrama esquemático. Verilog reconoce 12 compuertas básicas como primitivas predefinidas. Cuatro compuertas primitivas son del tipo de tres estados. Las otras ocho son las que se presentaron en la sección 2-7, y se declaran con las palabras clave en minúsculas **and, nand, or, nor, xor, xnor, not** y **buf.** Cuando se simulan

Tabla 4-9
Tablas de verdad para las compuertas primitivas predefinidas

and	0	1	x	z
0	0	0	0	0
1	0	1	x	x
x	0	x	x	x
z	0	x	x	x

or	0	1	x	z
0	0	1	x	x
1	1	1	1	1
x	x	1	x	x
z	x	1	x	x

xor	0	1	x	z
0	0	1	x	x
1	1	0	x	x
x	x	x	x	x
z	x	x	x	x

not	Entrada	Salida
	0	1
	1	0
	x	x
	z	x

las compuertas, el sistema asigna un conjunto lógico de cuatro valores a cada compuerta. Además de los dos valores lógicos 0 y 1, hay otros dos valores, *desconocido* y *alta impedancia*. Un valor desconocido se denota con **x,** y uno de alta impedancia, con **z.** Los valores desconocidos se consideran durante la simulación para el caso en que una entrada o salida es ambigua, por ejemplo, si todavía no se le ha asignado un valor de 0 o 1. La condición de alta impedancia se da en la salida de las compuertas de tres estados o si por descuido un alambre se deja desconectado. Las tablas de verdad de **and, or, xor** y **not** aparecen en la tabla 4-9. Las tablas de verdad para las demás compuertas son iguales, salvo que las salidas se complementan. Observe que, para la compuerta **and,** la salida es 1 sólo cuando ambas entradas son 1; la salida es 0 si cualquier entrada es 0. Por otra parte, si una entrada es **x** o **z,** la salida es **x.** La salida de la compuerta **or** es 0 si ambas entradas son 0, 1 si cualquier entrada es 1, y **x** en los demás casos.

Cuando una compuerta primitiva se incorpora en un módulo, decimos que se *crea un ejemplar* en el módulo. En general, los enunciados que crean ejemplares de componentes hacen referencia a componentes de nivel más bajo del diseño, con lo que básicamente se crean copias únicas (*ejemplares*) de esos componentes en el módulo de nivel más alto. Así, cuando un módulo usa una compuerta en su descripción decimos que *crea un ejemplar de* la compuerta.

A continuación se presentan dos ejemplos de modelado en el nivel de compuertas. Ambos ejemplos usan grupos de varios bits llamados *vectores*. Los vectores se especifican entre corchetes, con dos números separados por un signo de dos puntos. El código siguiente especifica dos vectores:

```
output [0:3]D;
wire [7:0]SUM;
```

El primer enunciado declara un vector de salida *D* de cuatro bits, 0 a 3. El segundo declara un vector alambrado *SUM* de ocho bits, numerados del 7 al 0. El primer número corresponde al bit más significativo del vector. Los bits individuales se especifican entre corchetes; por ejemplo *D*[2] especifica el bit 2 de *D*. También es posible direccionar partes de vectores. Por ejemplo, *SUM*[2:0] especifica los tres bits menos significativos del vector *SUM*.

El ejemplo HDL 4-1 es ilustrativo de la descripción en el nivel de compuertas de un decodificador de 2 a 4 líneas. Posee dos entradas de datos *A* y *B* y una entrada de habilitación *E*. Las cuatro salidas se especifican con el vector *D*. La declaración **wire** es para conexiones internas. Tres compuertas **not** producen el complemento de las entradas y cuatro compuertas **nand** generan las salidas para *D*. Recuerde que la salida siempre es el primer elemento en la lista de una compuerta, seguida de las entradas. Este ejemplo describe el decodificador de la fi-

Ejemplo HDL 4-1

```
//Descripción a nivel de compuertas del decodificador 2 a 4
//de la figura 4-19
module decoder_gl (A,B,E,D);
   input A,B,E;
   output [0:3]D;
   wire Anot,Bnot,Enot;
   not
      n1 (Anot,A),
      n2 (Bnot,B),
      n3 (Enot,E);
   nand
      n4 (D[0],Anot,Bnot,Enot),
      n5 (D[1],Anot,B,Enot),
      n6 (D[2],A,Bnot,Enot),
      n7 (D[3],A,B,Enot);
endmodule
```

gura 4-19 y sigue los procedimientos establecidos en la sección 3-9. Advierta que las palabras clave **not** y **nand** se escriben una sola vez y no tienen que repetirse para cada compuerta, pero hay que insertar comas al final de cada serie de compuertas con la excepción del último enunciado, que debe terminar con un punto y coma.

Es posible combinar dos o más módulos para construir una descripción jérárquica de un diseño. Existen dos tipos básicos de metodologías de diseño: descendente y ascendente. En un diseño *descendente*, se define el bloque de más alto nivel y luego se identifican los sub-bloques requeridos para construir el bloque de más alto nivel. En un diseño *ascendente*, primero se identifican los bloques de construcción y luego se combinan para construir el bloque de más alto nivel. Tomemos como ejemplo el sumador binario de la figura 4-9. Podemos considerarlo como un componente de bloque superior construido con cuatro bloques de sumador completo, cada uno de los cuales se construye con dos bloques de semisumador. En un diseño descendente, primero se define un sumador de cuatro bits y luego se describen el sumador completo y el semisumador. En un diseño ascendente, se define el semisumador, luego se construye el sumador completo y al último se construye el sumador de cuatro bits con los sumadores completos.

En el ejemplo HDL 4-2 se muestra una descripción jérárquica ascendente de un sumador de cuatro bits. El semisumador se define con ejemplares de compuertas primitivas. El siguiente módulo describe el sumador completo con dos ejemplares de semisumador. El tercer módulo describe el sumador de cuatro bits con cuatro ejemplares de sumador completo. (Tenga presente que los identificadores no pueden comenzar con un número, pero sí con un subraya, así que el nombre del módulo es `_4bit_adder`.) Los ejemplares se crean utilizando el nombre del módulo ejemplarizado con un conjunto nuevo de nombres de puerto (o el mismo). Por ejemplo, el ejemplar del semisumador HA1 dentro del módulo del sumador completo se crea con los puertos $S1$, $D1$, x, y. Esto produce un semisumador con las salidas $S1$, $D1$ y las entradas x, y.

Cabe señalar que, en Verilog, no es posible colocar una definición de módulo dentro de otra. En otras palabras, no está permitido insertar un módulo entre las palabras clave **module** y **endmodule** de otro módulo. La única forma de incorporar una definición de módulo en otro módulo es creando un ejemplar del mismo. Así, se crean ejemplares de módulos dentro de otros módulos para crear una descripción jérárquica de un diseño. Además, tome nota de que hay que especificar nombres al crear ejemplares de módulos ya definidos (como FA0 para el

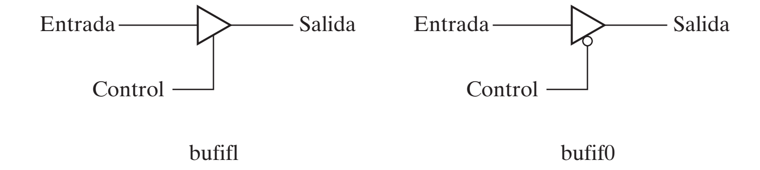
Ejemplo HDL 4-2

```
//Descripción jerárquica a nivel de compuertas de un sumador de 4 bits
//Descripción del semisumador (véase la figura 4-5b)
module halfadder (S,C,x,y);
   input x,y;
   output S,C;
//Crear ejemplares de compuertas primitivas
   xor (S,x,y);
   and (C,x,y);
endmodule

//Descripción de sumador completo (véase la figura 4-8)
module fulladder (S,C,x,y,z);
   input x,y,z;
   output S,C;
   wire S1,D1,D2; //Salidas de primera XOR y dos compuertas AND
//Crear un ejemplar del semisumador
    halfadder HA1 (S1,D1,x,y),
              HA2 (S,D2,S1,z);
    or g1(C,D2,D1);
endmodule

//Descripción del sumador de 4 bits (véase la figura 4-9)
module _4bit_adder (S,C4,A,B,C0);
   input [3:0] A,B;
   input C0;
   output [3:0] S;
   output C4;
   wire C1,C2,C3;  //Acarreos intermedios
//Crear un ejemplar del sumador completo
   fulladder  FA0 (S[0],C1,A[0],B[0],C0),
              FA1 (S[1],C2,A[1],B[1],C1),
              FA2 (S[2],C3,A[2],B[2],C2),
              FA3 (S[3],C4,A[3],B[3],C3);
endmodule
```

primer sumador completo en el tercer módulo), pero el uso de nombres es opcional al crear ejemplares de compuertas primitivas.

Compuertas de tres estados

Como se mencionó en la sección 4-10, las compuertas de tres estados tienen una entrada de control que puede colocar a la compuerta en un estado de alta impedancia. Dicho estado se indica con **z** en HDL. Hay cuatro tipos de compuertas de tres estados, que se ilustran en la figura 4-31. La compuerta **bufif1** se comporta como un búfer normal si control = 1. La salida pasa a un estado de alta impedancia **z** cuando control = 0. La compuerta **bufif0** se comporta de



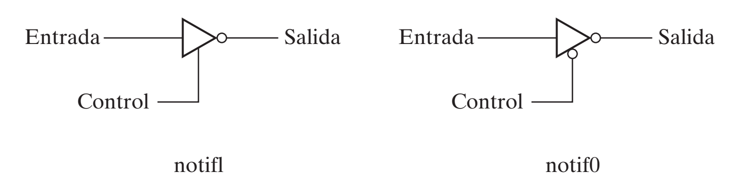


FIGURA 4-31
Compuertas de tres estados

forma similar excepto que el estado de alta impedancia se da cuando control = 1. Las dos compuertas **not** operan de forma similar, excepto que la salida es el complemento de la entrada cuando la compuerta no está en un estado de alta impedancia. Se crean ejemplares de las compuertas con el enunciado

nombre_compuerta (salida, entrada, control);

El nombre de la compuerta puede ser cualquiera de las cuatro compuertas de tres estados. La salida puede dar 0, 1 o **z.** Dos ejemplos de creación de ejemplares de compuertas son

```
bufif1  (OUT,A,control);
notif0  (Y,B,enable);
```

En el primer ejemplo, la entrada A se transfiere a OUT cuando control = 1. OUT pasa a **z** cuando control = 0. En el segundo ejemplo, la salida Y = **z** cuando enable = 1, y $Y = B'$ cuando enable = 0.

Las salidas de las compuertas de tres estados se pueden conectar entre sí para formar una línea de salida común. Para identificar semejante conexión, HDL usa la palabra clave **tri** de (triestado) para indicar que la salida tiene múltiples alimentaciones. Por ejemplo, consideremos el multiplexor de 2 líneas a 1 con compuertas de tres estados que se aprecia en la figura 4-32.

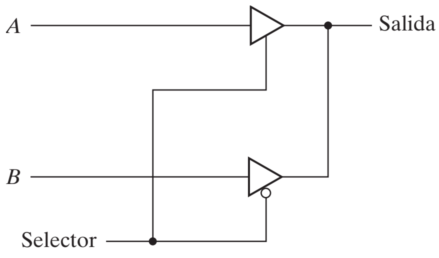


FIGURA 4-32
Multiplexor de 2 líneas a 1 con búferes de tres estados

La descripción en HDL debe usar el tipo de datos **tri** para la salida.

```
module muxtri (A,B,select,OUT);
   input A,B,select;
   output OUT;
   tri OUT;
   bufif1 (OUT,A,select);
   bufif0 (OUT,B,select);
endmodule
```

Los dos búferes de tres estados tienen la misma salida. Para indicar que existe una conexión común, es necesario declarar *OUT* con la palabra clave **tri.**

Las palabras clave **wire** y **tri** son ejemplos del tipo de datos *net*. Las redes (*nets*) representan conexiones entre elementos de hardware. Su valor depende continuamente de la salida del dispositivo que representan. La palabra *net* no es una palabra clave, pero representa una clase de tipos de datos como **wire, wor, wand, tri, supply1** y **supply0.** La declaración **wire** es la que se usa con mayor frecuencia. La red **wor** modela la implementación en hardware de la configuración OR alambrada. La **wand** modela la configuración AND alambrada (véase la figura 3-28). Las redes **supply1** y **supply0** representan fuente de poder y tierra. Se utilizan en el modelado a nivel de interruptores (véase la sección 10-10).

Modelado de flujo de datos

El modelado de flujo de datos utiliza varios operadores que actúan sobre operandos para producir resultados deseados. Verilog HDL cuenta con cerca de 30 tipos de operadores. En la tabla 4-10 se presentan algunos de ellos, su símbolo y la operación que realizan. (En la tabla 8-1, sección 8-2, se da una lista completa de operadores.) Hay que distinguir entre las operaciones aritméticas y lógicas, por lo que se usan símbolos distintos para cada una. El símbolo más (+) se utiliza para la suma aritmética, y el AND lógico emplea el símbolo &. Hay símbolos especiales para OR, NOT y XOR. El símbolo de igualdad es doble (sin espacio intermedio) para distinguirlo del que se usa en el enunciado de asignación. El operador de concatenación permite juntar varios operandos. Por ejemplo, es posible concatenar dos operandos de dos bits cada uno para formar un operando de cuatro bits. El operador condicional se explicará más adelante junto con el ejemplo HDL 4-6.

Tabla 4-10
Operadores de Verilog HDL

Símbolo	Operación
+	Suma binaria
−	Resta binaria
&	AND bit por bit
|	OR bit por bit
^	XOR bit por bit
~	NOT bit por bit
==	Igualdad
>	Mayor que
<	Menor que
{ }	Concatenación
? :	Condicional

Ejemplo HDL 4-3

```
//Descripción de flujo de datos del decodificador de 2 a 4
//Véase la figura 4-9
module decoder_df (A,B,E,D);
   input A,B,E;
   output [0:3] D;
   assign D[0] = ~(~A & ~B & ~E),
          D[1] = ~(~A & B & ~E),
          D[2] = ~(A & ~B & ~E),
          D[3] = ~(A & B & ~E);
endmodule
```

El modelado de flujo de datos utiliza asignaciones continuas y la palabra clave **assign.** Una asignación continua es un enunciado que asigna un valor a una *net*. El tipo de datos *net* se usa en Verilog HDL para representar una conexión física entre elementos de circuito. Una *net* define una salida de compuerta declarada por un enunciado **output** o **wire.** El valor asignado a la *net* se especifica con una expresión que utiliza operandos y operadores. Por ejemplo, suponiendo que ya se declararon las variables, un multiplexor de 2 líneas a 1 con entradas de datos *A* y *B*, entrada de selección *S* y salida *Y* se describe con la asignación continua

```
assign Y = (A & S) | (B & ~S);
```

Primero aparece la palabra clave **assign** seguida de la salida deseada *Y* y un signo de igual. Después viene una expresión booleana. En términos de hardware, esto equivaldría a conectar la salida de la compuerta OR (|) al alambre *Y*.

Los dos ejemplos que siguen muestran los modelos de flujo de datos de los dos ejemplos anteriores a nivel de compuertas. En el ejemplo HDL 4-3 se presenta la descripción de flujo de datos de un decodificador de 2 a 4 líneas. El circuito se define con cuatro enunciados de asignación continua empleando expresiones booleanas, una para cada salida. La descripción de flujo de datos del sumador de cuatro bits aparece en el ejemplo HDL 4-4. La lógica de suma se describe con un solo enunciado que usa los operadores de suma y concatenación. El símbolo más (+) especifica la suma binaria de los cuatro bits de *A* con los cuatro

Ejemplo HDL 4-4

```
//Descripción de flujo de datos de un sumador de 4 bits
module binary_adder (A,B,Cin,SUM,Cout);
   input [3:0] A,B;
   input Cin;
   output [3:0] SUM;
   output Cout;
   assign {Cout,SUM} = A + B + Cin;
endmodule
```

Ejemplo HDL 4-5

```
//Descripción de flujo de datos de un comparador de 4 bits
module magcomp (A,B,ALSB,AGTB,AEQB);
   input [3:0] A,B;
   output ALTB,AGTB,AEQB;
   assign ALTB=(A < B),
          AGTB = (A > B),
          AEQB = (A == B);
endmodule
```

bits de *B* y el bit de *Cin*. La salida deseada es la concatenación del acarreo de salida *Cout* y los cuatro bits de *SUM*. La concatenación de operandos se expresa entre llaves, separando los operandos con comas. Así, {`Cout, SUM`} representa el resultado de cinco bits de la operación de suma.

El modelado de flujo de datos permite describir circuitos combinacionales por su función en vez de por su estructura de compuertas. Para mostrar cómo las descripciones de flujo de datos facilitan el diseño digital, consideremos el comparador de magnitudes de cuatro bits que se describe en el ejemplo HDL 4-5. El módulo especifica dos entradas de cuatro bits, *A* y *B*, y tres salidas. Una salida (ALTB) es 1 lógico si *A* es menor que *B*; otra salida (AGTB) es 1 lógico si *A* es mayor que *B*; y una tercera salida (AEQB) es 1 lógico si *A* es igual a *B*. (Advierta que la igualdad se representa con dos símbolos de igual.) Un compilador de síntesis Verilog HDL puede aceptar como entrada esta descripción de módulo y producir la lista de un circuito equivalente a la figura 4-17.

El siguiente ejemplo usa el operador condicional (?:). Este operador acepta tres operandos:

condición ? expresión-verdadera : expresión-falsa ;

La condición se evalúa. Si el resultado es 1 lógico, se evalúa la expresión verdadera. Si el resultado es 0 lógico, se evalúa la expresión falsa. Esto equivale a una condición **if-else.** El ejemplo HDL 4-6 presenta la descripción de un multiplexor de 2 a 1 empleando el operador condicional. La asignación continua

```
assign OUT = select ? A : B ;
```

especifica la condición OUT $= A$ si select $= 1$, si select $= 0$, entonces OUT $= B$.

Ejemplo HDL 4-6

```
//Descripción de flujo de datos del multiplexor 2 a 1
module mux2x1_df (A,B,select,OUT);
   input A,B,select;
   output OUT;
   assign OUT = select ? A : B;
endmodule
```

Ejemplo HDL 4-7

```
//Descripción de comportamiento del multiplexor 2 a 1
module mux2x1_bh(A,B,select,OUT);
   input A,B,select;
   output OUT;
   reg OUT;
   always @ (select or A or B)
         if (select == 1) OUT = A;
         else OUT = B;
endmodule
```

Modelado de comportamiento

El modelado de comportamiento representa los circuitos digitales en un nivel funcional y algorítmico. Se le utiliza primordialmente para describir circuitos secuenciales, pero sirve también para describir circuitos combinacionales. Aquí se presentarán dos ejemplos sencillos de circuitos combinacionales como introducción al tema. Se analizará con mayor detalle el modelado de comportamiento en la sección 5-5, después de estudiar los circuitos secuenciales.

Las descripciones de comportamiento emplean la palabra clave **always** seguida de una lista de enunciados de asignación procedimentales. La salida deseada de los enunciados de asignación procedimentales debe ser del tipo de datos **reg.** A diferencia del tipo de datos **wire,** en el que la salida deseada de una asignación se puede actualizar continuamente, el tipo de datos **reg** conserva su valor hasta que se le asigna uno nuevo.

El ejemplo HDL 4-7 muestra la descripción de comportamiento de un multiplexor de 2 líneas a 1 (compare con el ejemplo HDL 4-6). Dado que la variable OUT es una salida deseada, debemos declararla como dato **reg** (además de la declaración **output**). Los enunciados de asignación procedimentales dentro del bloque **always** se ejecutan cada vez que hay un cambio en cualquiera de las variables indicadas después del símbolo @. (Observe que no se escribe un (;) al final del enunciado **always.**) En este caso, la lista incluye las variables de entrada *A* y *B*, y *select*. Advierta que se usa la palabra clave **or** entre las variables en lugar del operador de OR lógico "|". El enunciado condicional **if-else** permite tomar una decisión con base en el valor de la entrada *select*. El enunciado **if** se puede escribir sin el símbolo de igualdad:

```
if (select) OUT = A ;
```

Este enunciado implica que se examina *select* para ver si es 1 lógico.

El ejemplo HDL 4-8 describe la función de un multiplexor de 4 líneas a 1. La entrada *select* se define como un vector de dos bits, y la salida *y* se declara como dato **reg.** El enunciado **always** tiene un bloque secuencial delimitado por las palabras clave **case** y **endcase.** El bloque se ejecuta cada vez que cambia el valor de cualquiera de las entradas indicadas después del símbolo @. El enunciado **case** es una condición de ramificación condicional multivías. La expresión **case** (select) se evalúa y se compara con los valores de la lista de enunciados que siguen. Se ejecuta el primer valor que coincide con la condición verdadera. Puesto que *select* es un número de dos bits, puede ser igual a 00, 01, 10 o 11. Los números binarios se especifican con una **b** precedida por un apóstrofo. Primero se escribe el tamaño del número y luego su valor. Así, 2′b01 especifica un número binario de dos dígitos cuyo valor es 01. También pueden especificarse números en decimal, octal o hexadecimal, con las letras **′d, ′o** y **′h,** respectivamente. Si no se especifica la base del número, se toma como decimal por omisión. Si no se especifica el tamaño del número, el sistema supondrá que es de 32 bits.

Ejemplo HDL 4-8

```
//Descripción de comportamiento del multiplexor 4 a 1
//Describe la tabla de función de la figura 4-25b).
module mux4x1_bh (i0,i1,i2,i3,select,y);
   input i0,i1,i2,i3;
   input [1:0] select;
   output y;
   reg y;
   always @ (i0 or i1 or i2 or i3 or select)
            case (select)
               2'b00: y = i0;
               2'b01: y = i1;
               2'b10: y = i2;
               2'b11: y = i3;
            endcase
endmodule
```

Hemos mostrado aquí ejemplos sencillos de descripciones del comportamiento de circuitos combinacionales. El modelado de comportamiento y los enunciados de asignación procedimentales requieren conocimientos de circuitos secuenciales y se tratarán más a fondo en la sección 5-5.

Cómo escribir un conjunto de pruebas sencillo

Un conjunto de pruebas es un programa en HDL que sirve para aplicar estímulos a un diseño HDL, a fin de probarlo y observar su respuesta durante una simulación. En ocasiones, los conjuntos de pruebas resultan muy complejos y largos, y su desarrollo podría ser más tardado que el del diseño que se prueba. Sin embargo, los que se incluyen aquí son relativamente sencillos, ya que lo único que queremos probar es circuitos combinacionales. Presentamos los ejemplos para ilustrar descripciones representativas de módulos de estímulo HDL.

Además del enunciado **always,** los conjuntos de pruebas utilizan el enunciado **initial** para estimular el circuito probado. El enunciado **always** se ejecuta repetidamente en un ciclo. El enunciado **initial** sólo se ejecuta una vez en el tiempo = 0 de la simulación y podría continuar con cualesquier operaciones que se retarden en cierto número de unidades de tiempo, especificadas por el símbolo #. Por ejemplo, consideremos el bloque **initial**

```
initial
     begin
            A = 0; B= 0;
        #10 A = 1;
        #20 A = 0; B=1;
     end
```

El bloque se encierra entre las palabras clave **begin** y **end.** En el tiempo $= 0$, A y B se ponen en 0. Diez unidades de tiempo después, A se cambia a 1. Veinte unidades de tiempo después (en $t = 30$), A se cambia a 0 y B se cambia a 1. Las entradas de una tabla de verdad de tres bits se generan con el bloque **initial**

```
initial
    begin
            D = 3'b000;
            repeat (7)
        #10  D = D + 3'b001;
    end
```

El vector *D* de tres bits recibe el valor inicial 000 en el tiempo = 0. La palabra clave **repeat** especifica un enunciado cíclico: se suma 1 a *D* siete veces, una vez cada 10 unidades de tiempo. El resultado es una sucesión de números binarios de 000 a 111.

Un módulo de estímulo es un programa HDL que tiene la forma siguiente:

module nombreprueba.
Declarar identificadores locales **reg** y **wire.**
Crear ejemplares del módulo de diseño a probar.
Generar estímulos con enunciados **initial** y **always.**
Exhibir la respuesta de salida.
endmodule

Los módulos de prueba por lo regular carecen de entradas y de salidas. Las señales que se aplican como entradas al módulo de diseño simulado se declaran en el módulo de estímulo como del tipo de datos local **reg.** Las salidas del módulo de diseño que se exhiben para efectuar las pruebas se declaran en el módulo de estímulo como del tipo de datos local **wire.** Luego se crea un ejemplar del módulo a probar empleando los identificadores locales. La figura 4-33 aclara esta relación. El módulo de estímulo genera entradas para el módulo de diseño declarando los identificadores *TA* y *TB* como de tipo **reg,** y verifica la salida de la unidad de diseño con el identificador **wire** *TC*. Luego se usan los identificadores locales para crear el ejemplar del módulo de diseño a probar.

La respuesta al estímulo generado por los bloques **initial** y **always** aparecerá en la salida del simulador como diagramas de temporización. También es posible exhibir salidas numéricas empleando *tareas del sistema* de Verilog. Éstas son funciones integradas del sistema que se

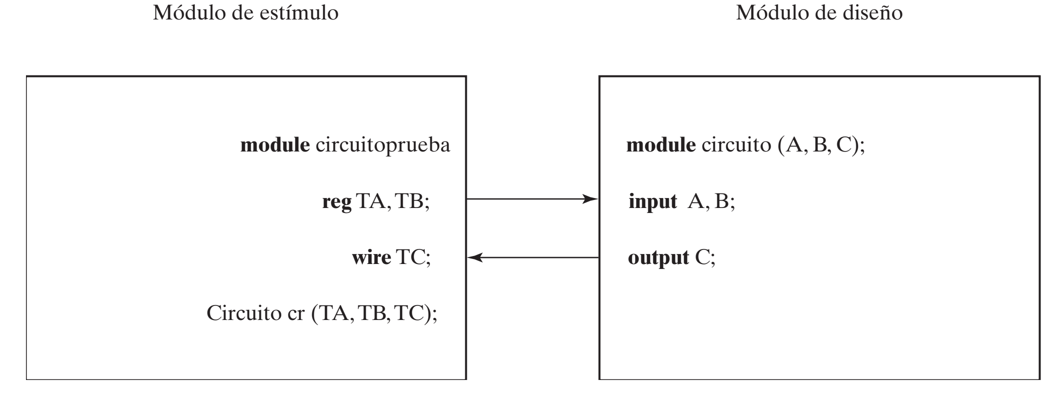


FIGURA 4-33
Interacción de los módulos de estímulo y de diseño

reconocen por palabras clave que inician con el símbolo $. Algunas de las tareas del sistema que sirven para exhibición son

$display—exhibir una vez el valor de variables o cadenas con un retorno de fin de línea,
$write**—igual que **$display pero sin pasar a la siguiente línea,
$monitor—exhibe variables cada vez que un valor cambia durante una simulación,
$time—muestra el tiempo de simulación,
$finish—termina la simulación.

La sintaxis de **$display, $write** y **$monitor** tiene la forma

Nombre-tarea (especificación de formato, lista argumentos);

La especificación de formato incluye la base de los números que se exhiben, lo que se indica con el símbolo (%), y podría tener una cadena encerrada entre comillas ("). La base puede ser binaria, decimal, hexadecimal u octal, lo que se indica con los símbolos %b, %d, %h y %o, respectivamente. Por ejemplo, el enunciado

```
$display (%d %b %b, C,A,B);
```

especifica la exhibición de *C* en decimal, y de *A* y *B* en binario. Observe que no hay comas en la especificación de formato, que la especificación de formato y la lista de argumentos se separan con una coma, y que la lista de argumentos lleva comas entre las variables. Un ejemplo de enunciado que especifica una cadena encerrada entre comillas es

$display ("tiempo = %0d A = %b B = %b", **$tiempo**,A,B);

que produce

```
tiempo = 3  A = 10  B = 1
```

donde (tiempo =), (A =) y (B =) forman parte de la cadena que se exhibirá. El formato %0d, %b y %b especifica la base de **$time,** *A* y *B*, respectivamente. Al exhibir valores de tiempo, es recomendable usar el formato %0d en lugar del formato %d. Esto exhibe los dígitos significativos sin los espacios a la derecha que se exhiben cuando se usa %d. (%d exhibe unos 10 espacios a la derecha porque el tiempo se calcula como un número de 32 bits.)

En el ejemplo HDL 4-9 se presenta un módulo de estímulo. El circuito a probar es el multiplexor 2 × 1 que se describió en el ejemplo 4-6. El módulo `testmux` no tiene puertos. Las entradas del multiplexor se declaran con la palabra clave **reg,** y las salidas, con la palabra clave **wire.** Se crea un ejemplar del multiplexor con las variables locales. El bloque **initial** especifica una sucesión de valores binarios que se aplicarán durante la simulación. La respuesta de salida se verifica con la tarea del sistema **$monitor.** Cada vez que una variable cambia de valor, el simulador exhibe las entradas, la salida y el tiempo. El resultado de la simulación aparece en la bitácora de simulación (simulation log) del ejemplo. Ahí vemos que $OUT = A$ cuando $S = 1$ y $OUT = B$ cuando $S = 0$, lo que verifica el funcionamiento del multiplexor.

La simulación lógica es un método rápido y exacto para analizar circuitos combinacionales y verificar que funcionan correctamente. Hay dos tipos de verificación: funcional y de tiempos. En la verificación *funcional*, estudiamos la operación lógica del circuito, independientemente de consideraciones de temporización. Esto se hace deduciendo la tabla de verdad del circuito combinacional. En la verificación *de tiempos*, estudiamos el funcionamiento del circuito incluyendo el efecto de los retardos en las compuertas. Esto se hace observando las formas de onda en las salidas de las compuertas cuando responden a una entrada dada. Presentamos un

Ejemplo HDL 4-9

```
//Estímulo para mux2x1_df.
module testmux;
  reg TA,TB,TS;  //entradas para mux
  wire Y;       //salida de mux
  mux2x1_df mx (TA,TB,TS,Y);  // crear un ejemplar mux
     initial
        begin
              TS = 1; TA = 0; TB = 1;
          #10 TA = 1; TB = 0;
          #10 TS = 0;
          #10 TA = 0; TB = 1;
        end
     initial
      $monitor("select = %b A = %b B = %b OUT = %b time = %0d",
               TS, TA, TB, Y, $time);
endmodule

//Descripción de flujo de datos de multiplexor de 2 a 1
//del ejemplo 4-6
module mux2x1_df (A,B,select,OUT);
   input A,B,select;
   output OUT;
   assign OUT = select ? A : B;
endmodule

Simulation log:

select = 1 A = 0 B = 1 OUT = 0 tiempo = 0
select = 1 A = 1 B = 0 OUT = 1 tiempo = 10
select = 0 A = 1 B = 0 OUT = 0 tiempo = 20
select = 0 A = 0 B = 1 OUT = 1 tiempo = 30
```

ejemplo de circuito con retardos de compuerta en la sección 3-9 (ejemplo HDL 3-3). Ahora presentaremos un ejemplo en HDL que produce la tabla de verdad de un circuito combinacional.

El análisis de circuitos combinacionales se explicó en la sección 4-2. Se analizó un circuito multinivel de un sumador completo y se dedujo su tabla de verdad por inspección. La descripción a nivel de compuertas de este circuito se ilustra en el ejemplo HDL 4-10. El circuito tiene tres entradas, dos salidas y nueve compuertas. La descripción del circuito sigue las interconexiones de las compuertas según el diagrama de la figura 4-2. El estímulo del circuito se da en el segundo módulo. Las entradas para estimular el circuito se especifican con un vector **reg** de tres bits llamado *D*. *D*[2] equivale a la entrada *A*, *D*[1] a la entrada *B* y *D*[0] a la entrada *C*. Las salidas del circuito, F_1 y F_2, se declaran como **wire.** Este procedimiento sigue los pasos bosquejados en la figura 4-33. El ciclo **repeat** proporciona los siete números binarios que siguen a 000, para la tabla de verdad. El resultado de la simulación genera la tabla de verdad que se incluye junto con el ejemplo. Esa tabla demuestra que el circuito es un sumador completo.

Ejemplo HDL 4-10

```
//Descripción a nivel de compuertas del circuito de la figura 4-2
module analysis (A,B,C,F1,F2);
   input   A,B,C;
   output  F1,F2;
   wire    T1,T2,T3,F2not,E1,E2,E3;
   or  g1 (T1,A,B,C);
   and g2 (T2,A,B,C);
   and g3 (E1,A,B);
   and g4 (E2,A,C);
   and g5 (E3,B,C);
   or  g6 (F2,E1,E2,E3);
   not g7 (F2not,F2);
   and g8 (T3,T1,F2not);
   or  g9 (F1,T2,T3);
endmodule

//Estímulo para analizar el circuito
module test_circuit;
  reg [2:0]D;
  wire F1,F2;
  analysis fig42(D[2],D[1],D[0],F1,F2);
  initial
    begin
     D = 3'b000;
     repeat(7)
      #10 D = D + 1'b1;
    end
  initial
     $monitor ("ABC = %b F1 = %b F2 =%b ",D, F1, F2);
endmodule


Simulation log:

ABC = 000 F1 = 0 F2 =0
ABC = 001 F1 = 1 F2 =0
ABC = 010 F1 = 1 F2 =0
ABC = 011 F1 = 0 F2 =1
ABC = 100 F1 = 1 F2 =0
ABC = 101 F1 = 0 F2 =1
ABC = 110 F1 = 0 F2 =1
ABC = 111 F1 = 1 F2 =1
```

PROBLEMAS

4-1 Considere el circuito combinacional de la figura P4-1.

a) Deduzca las expresiones booleanas para T_1 a T_4. Evalúe las salidas F_1 y F_2 en función de las cuatro entradas.

b) Escriba la tabla de verdad con 16 combinaciones binarias de las cuatro variables de entrada. Luego dé en la tabla los valores binarios de T_1 a T_4 y las salidas F_1 y F_2.

c) Grafique en mapas las funciones booleanas de salida obtenidas en la parte b) y demuestre que las expresiones booleanas simplificadas son equivalentes a las obtenidas en la parte a).

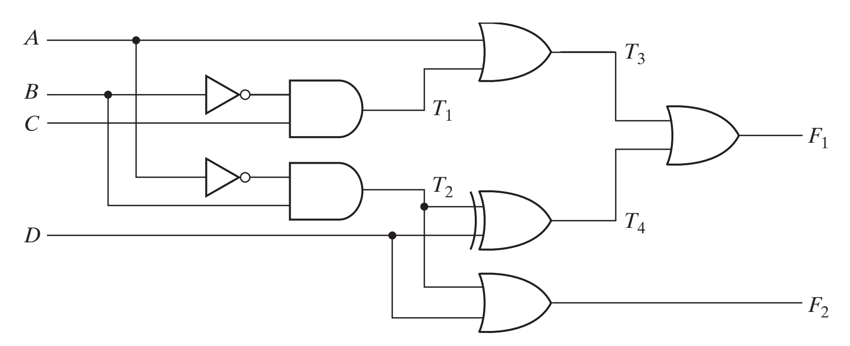


FIGURA P4-1

4-2 Obtenga las expresiones booleanas simplificadas para las salidas *F* y *G* en términos de las variables de entrada del circuito de la figura P4-2.

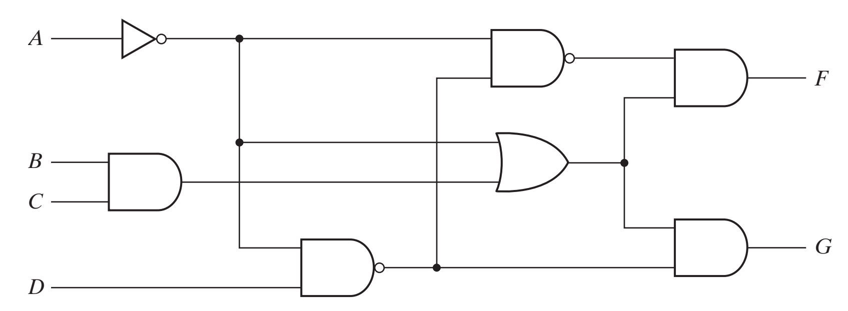


FIGURA P4-2

4-3 Para el circuito de la figura 4-26 (sección 4-10),

a) Escriba las funciones booleanas de las cuatro salidas en función de las variables de entrada.

b) Si el circuito se presenta como tabla de verdad, ¿cuántas filas y columnas tendría la tabla?

4-4 Diseñe un circuito combinacional con tres entradas y una salida. La salida es 1 cuando el valor binario de las entradas es menos que 3, y es 0 en los demás casos.

4-5 Diseñe un circuito combinacional con tres entradas, *x*, *y* y *z*, y tres salidas, *A*, *B* y *C*. Cuando la entrada binaria es 0, 1, 2 o 3, la salida binaria es uno más que la entrada. Si la entrada binaria es 4, 5, 6 o 7, la salida binaria es uno menos que la entrada.

4-6 Un circuito de mayoría es un circuito combinacional cuya salida es 1 si las variables de entrada tienen más unos que ceros. La salida es 0 en caso contrario. Diseñe un circuito de mayoría de tres entradas.

4-7 Diseñe un circuito combinacional que convierta un código Gray de cuatro bits (tabla 1-6) en un número binario de cuatro bits. Implemente el circuito con compuertas OR exclusivo.

4-8 Diseñe un convertidor de código que convierta un dígito decimal del código 8, 4, −2, −1 a BCD (véase la tabla 1-5).

4-9 Un decodificador de BCD a siete segmentos es un circuito combinacional que convierte un dígito decimal BCD en un código apropiado para seleccionar segmentos de un indicador que exhibe los dígitos decimales en la forma acostumbrada. Las siete salidas del decodificador (*a*, *b*, *c*, *d*, *e*, *f*, *g*) seleccionan los segmentos correspondientes del indicador, como se indica en la figura P4-9a). La forma de representar los dígitos decimales con el indicador se muestra en la figura P4-9b). Diseñe un decodificador de BCD a siete segmentos empleando el mínimo de compuertas. Las seis combinaciones no válidas deberán dejar el indicador en blanco.

a) Designación de segmentos

b) Designación numérica para exhibición

FIGURA P4-9

4-10 Diseñe un circuito combinacional complementador a dos, de cuatro bits. (La salida genera el complemento a dos del número binario de entrada.) Demuestre que es posible construir el circuito con compuertas OR exclusivo. ¿Puede predecir las funciones de salida para un complementador a dos de cinco bits?

4-11 Diseñe un circuito combinacional incrementador de cuatro bits. (Un circuito que suma 1 a un número binario de cuatro bits.) El circuito puede diseñarse con cuatro semisumadores.

4-12 a) Diseñe un circuito semirrestador con entradas x y y, y salidas D y B. El circuito resta los bits $x - y$, y coloca la diferencia en D y el préstamo (*borrow*) en B.

b) Diseñe un circuito restador completo con tres entradas, x, y y z, y dos salidas, D y B. El circuito resta $x - y - z$, donde z es el préstamo de entrada, B es el préstamo de salida y D es la diferencia.

4-13 El circuito sumador-restador de la figura 4-13 recibe los valores siguientes para la entrada de modo M y las entradas de datos A y B. En cada caso, determine los valores de las cuatro salidas *SUM*, el acarreo C y el desbordamiento V.

	M	A	B
a)	0	0111	0110
b)	0	1000	1001
c)	1	1100	1000
d)	1	0101	1010
e)	1	0000	0001

4-14 Suponga que la compuerta OR exclusivo tiene un retardo de propagación de 20 ns y que las compuertas AND y OR tienen un retardo de 10 ns. Calcule el retardo de propagación total del sumador de cuatro bits de la figura 4-12.

4-15 Deduzca la expresión booleana de dos niveles para el acarreo de salida C_4 que se muestra en el generador de acarreo anticipado de la figura 4-12.

4-16 Demuestre que es posible expresar el acarreo de salida de un circuito sumador completo en la forma AND-OR-INVERT

$$C_{i+1} = G_i + P_iC_i = \left(G_i'P_i' + G_i'C_i'\right)'$$

El CI tipo 74182 es un circuito generador de acarreo anticipado que genera los acarreos con compuertas AND-OR-INVERT (véase la sección 3-7). El circuito supone que las terminales de entrada tienen los complementos de las G, las P y C_1. Deduzca las funciones booleanas para los acarreos anticipados C_2, C_3 y C_4 en este CI. (*Sugerencia:* Use el método de sustitución de ecuaciones para deducir los acarreos en términos de C_1'.)

4-17 Defina el acarreo propagado y el acarreo generado como

$$P_i = A_i + B_i$$
$$G_i = A_iB_i$$

respectivamente. Demuestre que el acarreo de salida y la suma de salida de un sumador completo es

$$C_{i+1} = \left(C_i'G_i' + P_i'\right)'$$
$$S_i = \left(P_iG_i'\right) \oplus C_i$$

El diagrama lógico de la primera etapa de un sumador paralelo de cuatro bits como el implementado en el CI tipo 74283 se reproduce en la figura P4-17. Identifique las terminales P_i' y G_i' y demuestre que el circuito implementa un sumador completo.

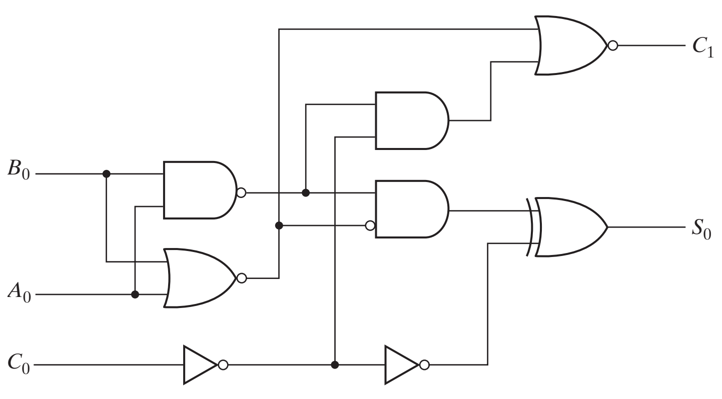


FIGURA P4-17
Primera etapa de un sumador paralelo

4-18 Diseñe un circuito combinacional que genere el complemento a nueve de un dígito BCD.

4-19 Construya un circuito sumador-restador BCD. Utilice el sumador BCD de la figura 4-14 y el complementador a nueve del problema 4-18. Utilice diagramas de bloque para los componentes.

4-20 Diseñe un multiplicador binario que multiplique dos números de cuatro bits. Utilice compuertas AND y sumadores binarios.

4-21 Diseñe un circuito combinacional que compare dos números de cuatro bits para ver si son iguales. La salida del circuito es 1 si los dos números son iguales, y 0 en caso contrario.

4-22 Diseñe un decodificador de exceso-3 a binario empleando las combinaciones no utilizadas del código como condiciones de indiferencia.

4-23 Dibuje el diagrama lógico de un decodificador de 2 a 4 líneas empleando únicamente compuertas NOR. Incluya una entrada de habilitación.

4-24 Diseñe un decodificador de BCD a decimal empleando las combinaciones no utilizadas del código BCD como condiciones de indiferencia.

4-25 Construya un decodificador de 5 a 32 líneas con cuatro decodificadores de 3 a 8 líneas provistos de habilitación y un decodificador de 2 a 4 líneas. Use diagramas de bloque para los componentes.

4-26 Construya un decodificador de 4 a 16 líneas con cinco decodificadores de 2 a 4 líneas provistos de habilitación.

4-27 Se especifica un circuito combinacional con estas tres funciones booleanas:

$$F_1(A, B, C) = \sum(2, 4, 7)$$
$$F_2(A, B, C) = \sum(0, 3)$$
$$F_3(A, B, C) = \sum(0, 2, 3, 4, 7)$$

Implemente el circuito con un decodificador construido con compuertas NAND (similar a la figura 4-19) y compuertas NAND o AND conectadas a las salidas del decodificador. Utilice un diagrama de bloque para el decodificador. Use el mínimo de entradas en las compuertas externas.

4-28 Se define un circuito combinacional con las tres funciones booleanas siguientes:

$$F_1 = x'y'z' + xz$$
$$F_2 = xy'z' + x'y$$
$$F_3 = x'y'z + xy$$

Diseñe el circuito con un decodificador y compuertas externas.

4-29 Diseñe un codificador prioritario con las cuatro entradas de la tabla 4-8, pero asignando a la entrada D_0 la prioridad más alta, y a D_3, la más baja.

4-30 Especifique la tabla de verdad de un codificador prioritario de octal a binario. Incluya una salida V para indicar que al menos una de las entradas está presente. La entrada con el subíndice más alto tendrá prioridad. ¿Qué valor tendrán las cuatro salidas si las entradas D_5 y D_3 son 1 al mismo tiempo?

4-31 Construya un multiplexor 16×1 con dos multiplexores 8×1 y uno 2×1. Use diagramas de bloque.

4-32 Implemente la función booleana siguiente con un multiplexor:

$$F(A, B, C, D) = \sum(0, 1, 3, 4, 8, 9, 15)$$

4-33 Implemente un sumador completo con dos multiplexores 4×1.

4-34 Un multiplexor 8 × 1 tiene las entradas *A*, *B* y *C* conectadas a las entradas de selección S_2, S_1 y S_0, respectivamente. Las entradas de datos I_0 a I_7 son: $I_1 = I_2 = I_7 = 0$; $I_3 = I_5 = 1$; $I_0 = I_4 = D$; e $I_6 = D'$. Determine la función booleana que implementa el multiplexor.

4-35 Implemente la siguiente función booleana con un multiplexor 4 × 1 y compuertas externas. Conecte las entradas *A* y *B* a las líneas de selección. Los requisitos de entrada de las cuatro líneas de datos serán función de las variables *C* y *D*. Estos valores se obtienen expresando *F* en función de *C* y *D* para cada uno de los cuatro casos en que $AB = 00, 01, 10$ y 11. Podría ser necesario implementar estas funciones con compuertas externas.

$$F(A, B, C, D) = \sum(1, 3, 4, 11, 12, 13, 14, 15)$$

4-36 Escriba la descripción HDL en el nivel de compuertas del circuito codificador prioritario de la figura 4-23.

4-37 Escriba la descripción jérárquica HDL en el nivel de compuertas de un sumador-restador de cuatro bits para números binarios sin signo. El circuito es similar a la figura 4-13 pero sin la salida *V*. Se puede usar un ejemplar del sumador completo de cuatro bits que se describe en el ejemplo HDL 4-2.

4-38 Escriba la descripción HDL de flujo de datos de un multiplexor cuádruple de 2 líneas a 1 con habilitación (véase la figura 4-26).

4-39 Escriba una descripción HDL del comportamiento de un comparador de cuatro bits con una salida de seis bits *Y* [5 : 0]. El bit 5 de *Y* es para igualdad, el bit 4 para desigualdad, el bit 3 para mayor que, el bit 2 para menor que, el bit 1 para mayor o igual que, y el bit 0 para menor o igual que.

4-40 Escriba una descripción HDL de flujo de datos de un sumador-restador de números sin signo de cuatro bits. Utilice el operador condicional (?:).

4-41 Repita el problema 4-40 empleando modelado de comportamiento.

4-42 a) Escriba una descripción HDL en el nivel de compuertas del circuito convertidor de BCD a exceso-3 que se ilustra en la figura 4-4.
b) Escriba una descripción de flujo de datos del convertidor de BCD a exceso-3 utilizando las expresiones booleanas de la figura 4-3.
c) Escriba una descripción HDL del comportamiento de un convertidor de BCD a exceso-3.
d) Escriba un conjunto de pruebas para simular y probar el circuito convertidor de BCD a exceso-3 y verificar la tabla de verdad. Compruebe los tres circuitos.

4-43 Explique la función del circuito especificado por la descripción HDL siguiente:

```
module Prob438 (A,B,S,E,Q);
   input  [1:0] A, B;
   input  S, E;
   output [1:0] Q;
   assign Q = E ? (S ? A : B) : 'bz;
endmodule
```

4-44 Escriba una descripción HDL del comportamiento de una unidad de aritmética-lógica (ALU) de cuatro bits. El circuito efectúa dos operaciones aritméticas y dos lógicas que se seleccionan con una entrada de dos bits. Las cuatro operaciones son suma, resta, AND y OR.

4-45 Escriba una descripción HDL del comportamiento de un codificador prioritario de cuatro entradas. Use un vector de cuatro bits para las entradas *D* y un bloque **always** con enunciados **if-else.** Suponga que la entrada *D*[3] es prioritaria.

REFERENCIAS

1. DIETMEYER, D. L. 1988. *Logic Design of Digital Systems,* 3a. ed. Boston: Allyn Bacon.
2. GAJSKI, D. D. 1997. *Principles of Digital Design.* Upper Saddle River, NJ: Prentice-Hall.
3. HAYES, J. P. 1993. *Introduction to Digital Logic Design.* Reading, MA: Addison-Wesley.
4. KATZ, R. H. 1994. *Contemporary Logic Design.* Upper Saddle River, NJ: Prentice-Hall.
5. MANO, M. M. y C. R. KIME. 2000. *Logic and Computer Design Fundamentals,* 2a. ed. Upper Saddle River, NJ: Prentice-Hall.
6. NELSON V. P., H. T. NAGLE, J. D. IRWIN y B. D. CARROLL. 1995. *Digital Logic Circuit Analysis and Design.* Upper Saddle River, NJ: Prentice-Hall.
7. ROTH, C. H. 1992. *Fundamentals of Logic Design,* 4a. ed. St. Paul: West.
8. WAKERLY, J. F. 2000. *Digital Design: Principles and Practices,* 3a. ed. Upper Saddle River, NJ: Prentice-Hall.
9. BHASKER, J. 1997. A *Verilog HDL Primer.* Allentown, PA: Star Galaxy Press.
10. BHASKER, J. 1998. *Verilog HDL Synthesis.* Allentown, PA: Star Galaxy Press.
11. CILETTI, M. D. 1999. *Modeling, Synthesis, and Rapid Prototyping with Verilog HDL.* Upper Saddle River, NJ: Prentice-Hall.
12. PALNITKAR, S. 1996. *Verilog HDL: A Guide to Digital Design and Synthesis.* SunSoft Press (un título Prentice-Hall).
13. THOMAS, D. E. y P. R. MOORBY. 1998. *The Verilog Hardware Description Language,* 4a. ed. Boston: Kluwer Academic Publishers.

5 Lógica secuencial sincrónica

5-1 CIRCUITOS SECUENCIALES

Los circuitos digitales estudiados hasta ahora han sido combinacionales: sus salidas dependen exclusivamente de las entradas actuales. Aunque es probable que todos los sistemas digitales tengan circuitos combinacionales, casi todos los que se usan en la práctica también incluyen elementos de almacenamiento, que requieren que el sistema se describa en términos de *lógica secuencial*.

En la figura 5-1 se presenta un diagrama de bloques de un circuito secuencial. Consiste en un circuito combinacional al que se conectan elementos de almacenamiento para formar una trayectoria de retroalimentación. Los elementos de almacenamiento son dispositivos capaces de guardar información binaria. La información almacenada en estos elementos en cualquier momento dado define el *estado* del circuito secuencial en ese momento. El circuito secuencial recibe información binaria de entradas externas. Esas entradas, junto con el estado actual de los elementos de almacenamiento, determinan el valor binario de las salidas. También determinan la condición para cambiar el estado de los elementos de almacenamiento. El diagrama de bloques indica que las salidas de un circuito secuencial son función no sólo de las entradas, sino también del estado actual de los elementos de almacenamiento. El siguiente estado de los elementos de almacenamiento también es función de entradas externas y del estado actual. Así pues,

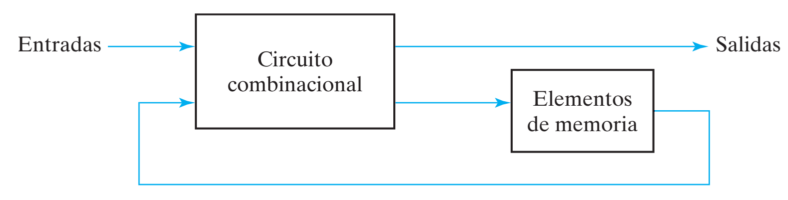


FIGURA 5-1
Diagrama de bloques de un circuito secuencial

un circuito secuencial se especifica con una sucesión temporal de entradas, salidas y estados internos.

Hay dos tipos principales de circuitos secuenciales, y su clasificación depende de los tiempos de sus señales. Un circuito secuencial *sincrónico* es un sistema cuyo comportamiento se define conociendo sus señales en instantes discretos. El comportamiento de un circuito secuencial *asincrónico* depende de las señales de entrada en cualquier instante dado y del orden en que cambian las entradas. Los elementos de almacenamiento que suelen usarse en los circuitos secuenciales asincrónicos son dispositivos de retardo de tiempo. La capacidad de almacenamiento de un dispositivo de retardo de tiempo se debe al tiempo que la señal tarda en propagarse por el dispositivo. En la práctica, el retardo interno de propagación de las compuertas lógicas tiene la suficiente duración como para producir el retardo requerido, de modo que podrían no ser necesarias unidades de retardo adicionales. En los sistemas asincrónicos tipo compuerta, los elementos de almacenamiento consisten en compuertas lógicas cuyo retardo de propagación hace posible el almacenamiento requerido. Así, un circuito secuencial asincrónico podría considerarse como un circuito combinacional con retroalimentación. Gracias a la retroalimentación entre compuertas lógicas, el circuito secuencial asincrónico podría volverse inestable ocasionalmente. El problema de inestabilidad impone muchas dificultades al diseñador. Los circuitos secuenciales asincrónicos se estudiarán en el capítulo 9.

Un circuito secuencial sincrónico utiliza señales que afectan a los elementos de almacenamiento únicamente en instantes discretos. La sincronización se logra con un dispositivo de temporización llamado *generador de reloj*, el cual produce un tren periódico de *pulsos de reloj*. Los pulsos de reloj se distribuyen por todo el sistema de modo que los elementos de almacenamiento sólo se vean afectados al llegar cada pulso. En la práctica, los pulsos de reloj se aplican con otras señales que especifican el cambio requerido en los elementos de almacenamiento. Los circuitos secuenciales sincrónicos que usan pulsos de reloj en las entradas de sus elementos de almacenamiento se denominan *circuitos secuenciales con reloj*, y son el tipo que se usa más comúnmente en la práctica. Casi nunca manifiestan problemas de estabilidad y es fácil dividir su temporización en pasos discretos independientes, cada uno de los cuales se puede considerar por separado.

Los elementos de almacenamiento empleados en los circuitos secuenciales con reloj se llaman *flip-flops*. Un flip-flop es un dispositivo binario de almacenamiento que puede almacenar un bit de información. Un circuito secuencial podría usar muchos flip-flops para almacenar tantos bits como sea necesario. En la figura 5-2 se ilustra el diagrama de bloques de un circuito secuencial

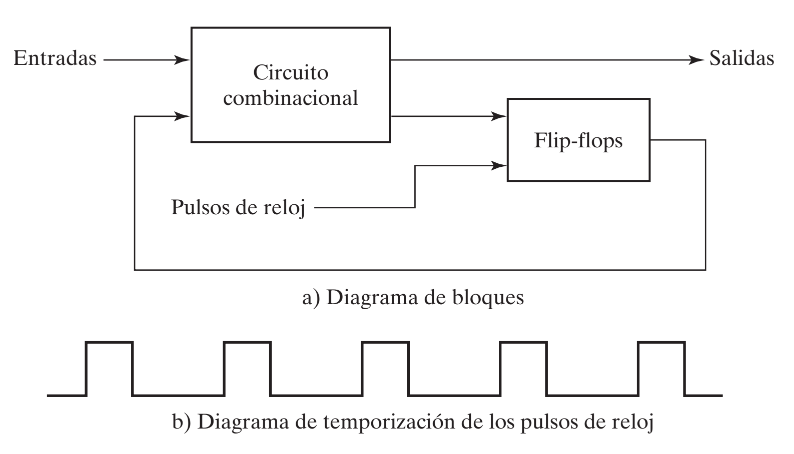


FIGURA 5-2
Circuito secuencial sincrónico con reloj

sincrónico con reloj. Las salidas pueden provenir del circuito combinacional o de los flip-flops, o de ambos. Los flip-flops reciben sus entradas del circuito combinacional y también de una señal de reloj cuyos pulsos se presentan a intervalos fijos de tiempo, como se observa en el diagrama de temporización. El estado del flip-flop sólo puede cambiar durante una transición de pulso de reloj. Cuando el pulso de reloj no está activo, el ciclo de retroalimentación se rompe porque las salidas del flip-flop no pueden cambiar aunque cambie el valor de las salidas del circuito combinacional que alimenta sus entradas. Por tanto, la transición de un estado al siguiente se da únicamente a intervalos de tiempo preestablecidos, dictados por los pulsos de reloj.

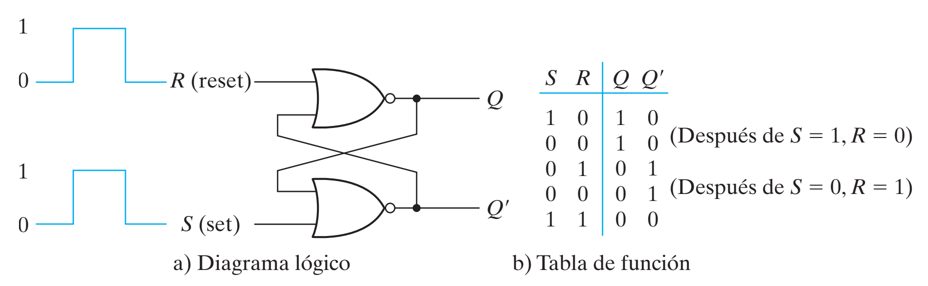
5-2 LATCHES

Un circuito flip-flop puede mantener un estado binario indefinidamente (en tanto se alimente electricidad al circuito), hasta que una señal de entrada le indique que debe cambiar de estado. Las principales diferencias entre los diversos tipos de flip-flops radican en el número de entradas que tienen y en la forma en que las entradas afectan el estado binario. Los tipos más básicos de flip-flops operan con niveles de señal y se llaman *latches*. Los latches que presentaremos aquí son los circuitos básicos con los que se construyen todos los flip-flops. Aunque los latches son útiles para almacenar información binaria y para diseñar circuitos secuenciales asincrónicos (véase la sección 9-3), no resultan prácticos en los circuitos secuenciales sincrónicos. En la sección que sigue presentaremos los tipos de flip-flops que se usan en los circuitos secuenciales.

Latch *SR*

El latch *SR* es un circuito con dos compuertas NOR acopladas en cruz o dos compuertas NAND acopladas en cruz. Tiene dos entradas, S (de *set*, establecer) y R (de *reset*, restablecer). El latch *SR* que se construye con dos compuertas NOR acopladas en cruz se aprecia en la figura 5-3. El latch tiene dos estados útiles. Cuando las salidas $Q = 1$ y $Q' = 0$, decimos que está en el *estado establecido*. Cuando $Q = 0$ y $Q' = 1$, está en el *estado restablecido*. Las salidas Q y Q' normalmente son una el complemento de la otra, pero si ambas entradas son 1 al mismo tiempo, se presenta un estado indefinido en el que ambas salidas son 0.

En condiciones normales, las dos entradas del latch permanecen en 0 a menos que se deba cambiar de estado. La aplicación momentánea de un 1 a la entrada S hace que el latch pase al estado establecido. La entrada S debe volver a 0 antes de cualquier otro cambio, para que no se presente el estado indefinido. Como se indica en la tabla de función de la figura 5-3b), dos condiciones de entrada hacen que el circuito esté en el estado establecido. La primera condición ($S = 1$,



S	R	Q	Q'	
1	0	1	0	
0	0	1	0	(Después de $S = 1, R = 0$)
0	1	0	1	
0	0	0	1	(Después de $S = 0, R = 1$)
1	1	0	0	

b) Tabla de función

FIGURA 5-3
Latch *SR* con compuertas NOR

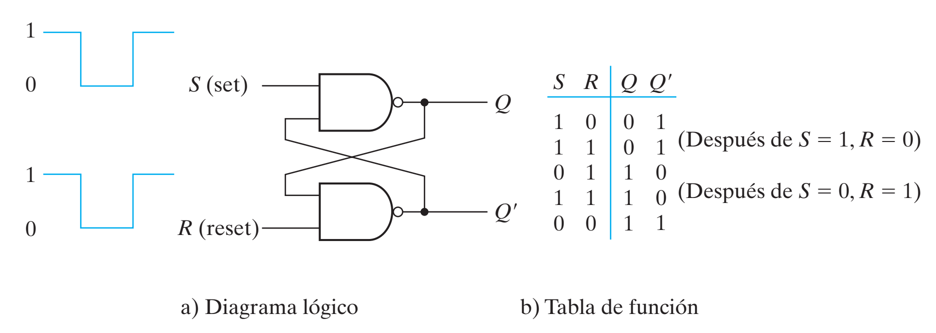
$R = 0$) es la acción que debe efectuar la entrada S para poner el circuito en el estado establecido. Cuando la entrada activa se quita de S, el circuito permanece en el mismo estado. Una vez que las dos entradas regresan a 0, será posible cambiar al estado restablecido aplicando momentáneamente un 1 a la entrada R. Luego puede quitarse el 1 de R y el circuito permanecerá en el estado restablecido. Así pues, cuando ambas entradas, S y R, son 0, el latch estará en el estado establecido o en el restablecido, dependiendo de cuál entrada fue 1 más recientemente.

Si se aplica un 1 a las dos entradas S y R del latch, ambas salidas cambian a 0. Esto produce un estado indefinido porque se hace imposible predecir cuál será el siguiente estado cuando ambas entradas vuelvan a 0, y también viola el requisito de que una salida sea el complemento de la otra. Durante el funcionamiento normal, esta condición se evita asegurándose de que no se aplique 1 a ambas entradas simultáneamente.

El latch *SR* con dos compuertas NAND acopladas en cruz se muestra en la figura 5-4. Opera con ambas entradas normalmente en 1, a menos que sea preciso cambiar el estado del latch. La aplicación de 0 a la entrada S hace que la salida Q cambie a 1 y coloca al latch en el estado establecido. Cuando la salida S vuelve a 1, el circuito permanece en el estado establecido. Una vez que ambas entradas vuelven a 1, se permite cambiar el estado del latch aplicando un 0 a la entrada R. Esto hace que el circuito vuelva al estado restablecido y permanezca en él aun después de que ambas entradas vuelven a 1. La condición que no está definida en el caso del latch NAND es que ambas entradas sean 0 al mismo tiempo, así que debe evitarse esa combinación de entradas.

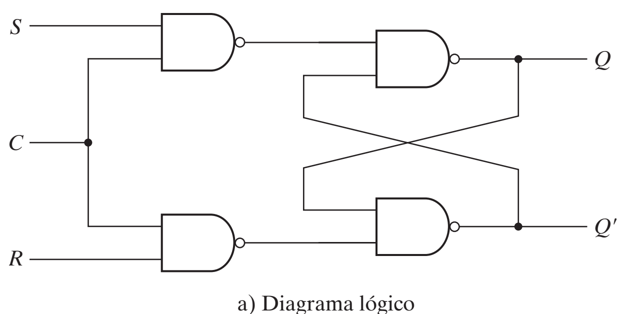
Si comparamos el latch NAND con el NOR, veremos que las señales de entrada del latch NAND requieren el complemento de los valores empleados para el latch NOR. Dado que el latch NAND requiere una señal 0 para cambiar su estado, también se le conoce como latch $S'R'$. Los apóstrofos (o testas sobre las letras) indican que las entradas deben estar en su forma complementada para activar el circuito.

Es posible modificar el funcionamiento del latch *SR* básico incluyendo una entrada de control adicional que determina cuándo puede cambiarse el estado del latch. En la figura 5-5 se muestra un latch *SR* con entrada de control. Consiste en el latch *SR* básico y dos compuertas NAND adicionales. La entrada de control C actúa como señal de habilitación para las otras dos entradas. La salida de las compuertas NAND permanecerán en el nivel 1 lógico en tanto la entrada de control permanezca en 0. Ésta es la condición latente del latch *SR*. Cuando la entrada de control cambia a 1, se permite que la información de la entrada S o R afecte al latch *SR*. Se alcanza el estado establecido con $S = 1$, $R = 0$ y $C = 1$. Para cambiar al estado restablecido, las entradas deben ser $S = 0$, $R = 1$ y $C = 1$. En ambos casos, cuando C regresa a 0, el cir-



S	R	Q	Q'	
1	0	0	1	
1	1	0	1	(Después de $S = 1, R = 0$)
0	1	1	0	
1	1	1	0	(Después de $S = 0, R = 1$)
0	0	1	1	

FIGURA 5-4
Latch *SR* con compuertas NAND



a) Diagrama lógico

C	S	R	Siguiente estado de Q
0	X	X	Sin cambio
1	0	0	Sin cambio
1	0	1	$Q = 0$; estado restablecido
1	1	0	$Q = 1$; estado establecido
1	1	1	Indeterminado

b) Tabla de función

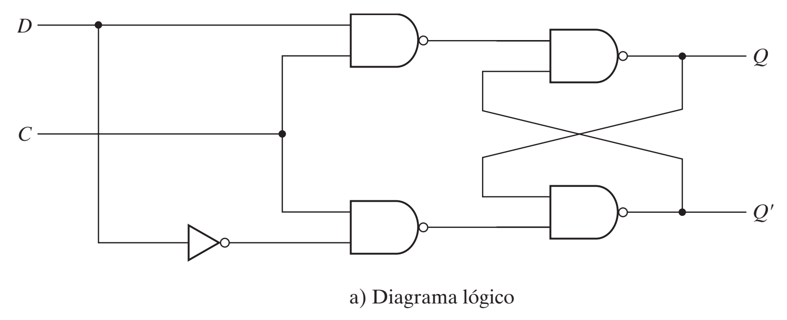
FIGURA 5-5
Latch *SR* con entrada de control

cuito permanece en su estado actual. La entrada de control inhabilita el circuito aplicando 0 a *C*, de modo que el estado de la salida no cambie sean cuales sean los valores de *S* y *R*. Además, cuando $C = 1$ y ambas entradas *S* y *R* son 0, el estado del circuito permanece sin cambio. Estas condiciones se presentan en la tabla de función que acompaña al diagrama.

Se presenta una condición indeterminada cuando las tres entradas son 1. Esta condición coloca ceros en ambas entradas del latch *SR* básico, lo que hace que éste pase al estado indefinido. Cuando la entrada de control vuelva a 0, no será posible determinar de forma concluyente el próximo estado, pues dependerá de cuál entrada, *S* o *R*, cambie primero a 0. Esta condición indeterminada dificulta el manejo de este circuito y casi nunca se usa en la práctica. No obstante, es un circuito importante porque otros latches y flip-flops se construyen con él.

Latch *D*

Una forma de eliminar la condición indeseable del estado indeterminado en el latch *SR* es garantizar que las entradas *S* y *R* nunca sean 1 al mismo tiempo. Esto se hace en el latch *D* que se ilustra en la figura 5-6. Este latch sólo tiene dos entradas: *D* (datos) y *C* (control). La entrada *D* pasa directamente a la entrada *S* y su complemento se aplica a la entrada *R*. En tanto la entrada de control esté en 0, el latch *SR* acoplado en cruz tendrá ambas entradas en el nivel 1 y el circuito no podrá cambiar de estado sea cual sea el valor de *D*. La entrada *D* se muestrea cuando $C = 1$. Si $D = 1$, la salida *Q* pasará a 1, colocando el circuito en el estado establecido. Si $D = 0$, la salida *Q* pasará a 0, colocando el circuito en el estado restablecido.



a) Diagrama lógico

C	D	Siguiente estado de Q
0	X	Sin cambio
1	0	$Q = 0$; estado restablecido
1	1	$Q = 1$; estado establecido

b) Tabla de función

FIGURA 5-6
Latch *D*

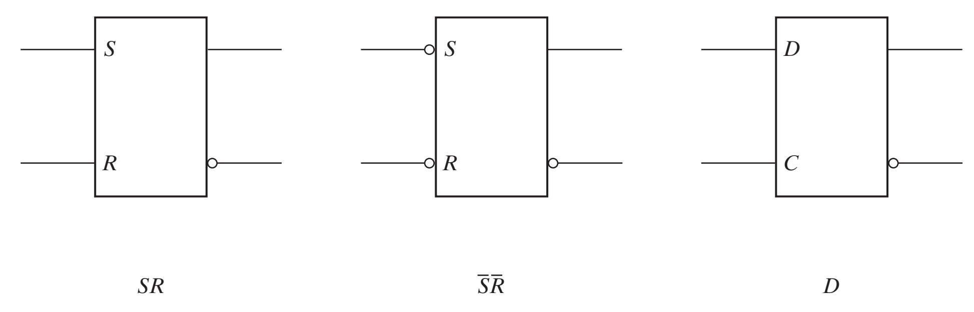


FIGURA 5-7
Símbolos gráficos de latches

El latch *D* se llama así por su capacidad para almacenar datos en su interior. Es apropiado para usarse como almacenamiento temporal de información binaria entre una unidad y su entorno. La información binaria presente en la entrada de datos del latch *D* se transfiere a la salida *Q* cuando se habilita la entrada de control. La salida seguirá los cambios en la entrada de datos en tanto esté habilitada la entrada de control. Esta situación crea un camino de la entrada *D* a la salida, y es por ello que el circuito se conoce como latch *transparente*. Cuando se inhabilita la entrada de control, la información binaria que estaba presente en la entrada de datos en el momento en que se presentó la transición se conservará en la salida *Q* hasta que se habilite otra vez la entrada de control.

En la figura 5-7 se presentan los símbolos gráficos para los distintos latches. Los latches se representan con un rectángulo cuyas entradas están a la izquierda, y sus salidas, a la derecha. Una salida indica la salida normal, y la otra (con burbuja), su complemento. En el símbolo gráfico del latch *SR*, las entradas *S* y *R* se indican dentro del rectángulo. En el caso de un latch de compuertas NAND, se añaden burbujas a las entradas para indicar que el establecimiento y el restablecimiento se efectúan con la señal de 0 lógico. En el símbolo gráfico del latch *D* las entradas *D* y *C* se indican dentro del rectángulo.

5-3 FLIP-FLOPS

El estado de un latch o flip-flop se conmuta con un cambio en la entrada de control. Este cambio momentáneo se denomina *disparo* y decimos que la transición que causa dispara el flip-flop. El latch *D* con pulsos en su entrada de control es básicamente un flip-flop que se dispara cada vez que el pulso alcanza el nivel de 1 lógico. En tanto la entrada de pulso se mantenga en este nivel, cualquier cambio en la entrada de datos hará que cambie la salida y el estado del latch.

Como se aprecia en el diagrama de bloques de la figura 5-2, un circuito secuencial tiene una trayectoria de retroalimentación de las salidas de los flip-flops a la entrada del circuito combinacional. Por tanto, las entradas de los flip-flops se derivan en parte de las salidas de esos mismos flip-flops y de otros. Cuando se usan latches como elementos de almacenamiento, surge una dificultad grave. Las transiciones de estado de los latches se inician tan pronto como el pulso de reloj cambia al nivel de 1 lógico. El nuevo estado del latch aparece en la salida mientras el pulso aún está activo. Esta salida se conecta a las entradas de los latches a través del circuito combinacional. Si las entradas aplicadas a los latches cambian mientras el pulso de reloj todavía está en el nivel de 1 lógico, los latches responderán a nuevos valores y podría presentarse un nuevo estado de salida. El resultado es una situación impredecible, ya que el estado de los latches podría seguir cambiando durante todo el tiempo que el pulso de reloj se mantiene en el

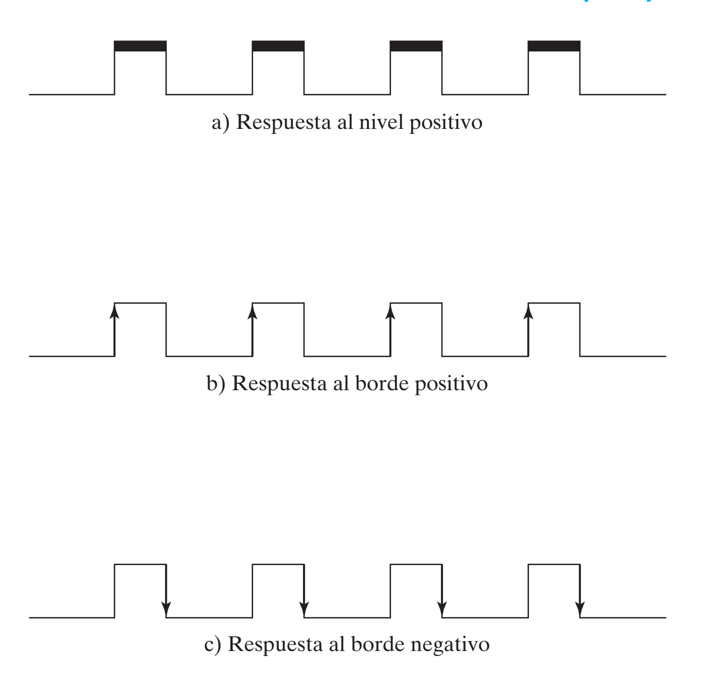


FIGURA 5-8
Respuesta al reloj en un latch y un flip-flop

estado activo. Debido a este funcionamiento poco confiable, no es posible aplicar directamente, ni a través de lógica combinacional, la salida de un latch a la entrada del mismo latch o de otro, cuando todos los latches se disparan con la misma fuente de reloj.

Los circuitos de flip-flop se construyen de tal manera que funcionan correctamente cuando forman parte de un circuito secuencial que utiliza un solo reloj. El problema del latch es que responde a un cambio en el *nivel* de un pulso de reloj. Como se observa en la figura 5-8a), una respuesta de nivel positivo en la entrada de control permite cambios en la salida cuando la entrada *D* cambia mientras el reloj se mantiene en el nivel de 1 lógico. La clave para que el flip-flop funcione correctamente es dispararlo únicamente durante una *transición* de la señal. Un pulso de reloj sufre dos transiciones: de 0 a 1 y de 1 a 0 al regresar. Como se aprecia en la figura 5-8, la transición positiva se define como el borde (o flanco) positivo, y la negativa, como el borde negativo. Hay dos formas de modificar un latch para formar un flip-flop. Una consiste en utilizar dos latches en una configuración especial que aísla la salida del flip-flop para que no se vea afectada mientras su entrada está cambiando. Otra consiste en producir un flip-flop que se dispare únicamente durante una transición de señal (de 0 a 1 o de 1 a 0) y quede inhabilitado durante el resto del pulso de reloj. Ahora mostraremos la implementación de ambos tipos de flip-flop.

Flip-flop *D* disparado por borde (o flanco)

En la figura 5-9 se representa la construcción de un flip-flop *D* con dos latches *D* y un inversor. El primer latch es el amo, y el segundo, el esclavo. El circuito muestrea la entrada *D* y cambia su salida *Q* únicamente en el borde negativo del reloj controlador (designado por *CLK* [*clock*]). Cuando el reloj es 0, la salida del inversor es 1. El latch esclavo queda habilitado y su salida *Q* es igual a la salida *Y* del amo. El latch amo queda inhabilitado porque $CLK = 0$.

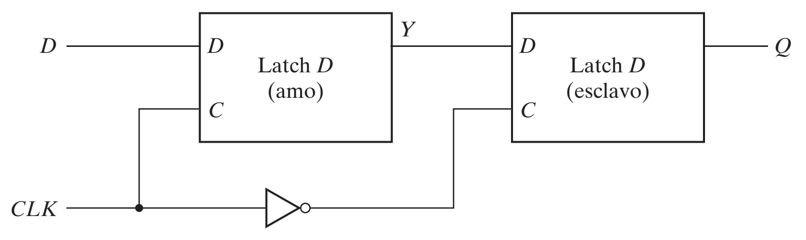


FIGURA 5-9
Flip-flop *D* amo-esclavo

Cuando el pulso de entrada cambia al nivel de 1 lógico, el dato de la entrada *D* externa se transfiere al amo. El esclavo, empero, estará inhabilitado en tanto el reloj permanezca en el nivel 1, porque su entrada *C* es 0. Cualquier cambio en la entrada hará que cambie la salida *Y* del amo, pero no afectará la salida del esclavo. Cuando el pulso vuelva a 0, el amo quedará inhabilitado y aislado de la entrada *D*. Al mismo tiempo, el esclavo estará habilitado y el valor *Y* se transferirá a la salida *Q* del flip-flop. Así, la salida del flip-flop sólo puede cambiar durante la transición del reloj de 1 a 0.

El comportamiento del flip-flop amo-esclavo que acabamos de describir implica que la salida sólo puede cambiar durante el borde negativo del reloj. También es posible diseñar el circuito de modo que la salida del flip-flop cambie en el borde positivo del reloj. Esto sucede en un flip-flop que tiene un inversor adicional entre la terminal *CLK* y la unión entre el otro inversor y la entrada *C* del latch amo. Un flip-flop semejante se dispara con un pulso negativo, de modo que el borde negativo del reloj afecte al amo, y el positivo, al esclavo y a la terminal de salida.

Una construcción más eficiente de un flip-flop *D* disparado por borde utiliza tres latches *SR*, como se muestra en la figura 5-10. Dos latches responden a las entradas externas *D* (datos) y

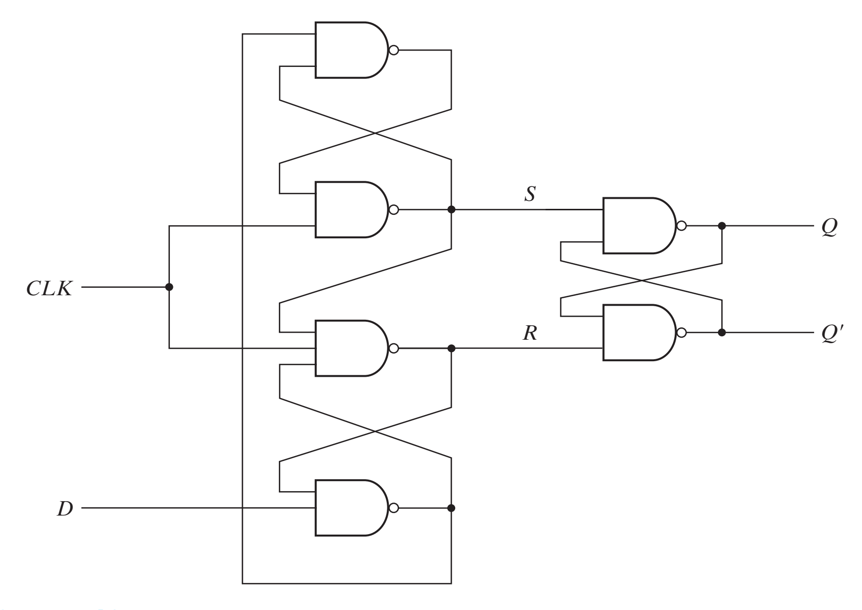


FIGURA 5-10
Flip-flop tipo *D* disparado por borde positivo

CLK (reloj). El tercer latch proporciona las salidas para el flip-flop. Las entradas *S* y *R* del latch de salida se mantienen en el nivel 1 lógico cuando $CLK = 0$. Esto hace que la salida permanezca en su estado actual. La entrada *D* podría ser 0 o 1. Si $D = 0$ cuando *CLK* pasa a 1, *R* cambia a 0. Esto hace que el flip-flop pase al estado restablecido, de modo que $Q = 0$. Si hay un cambio en la entrada *D* mientras $CLK = 1$, la terminal *R* permanecerá en 0. Así, el flip-flop queda bloqueado y no responde a más cambios en la entrada. Cuando el reloj vuelve a 0, *R* cambia a 1 y hace que el latch de salida pase a la condición latente sin cambiar su salida. Asimismo, si $D = 1$ cuando *CLK* pasa de 0 a 1, *S* cambiará a 0. Esto hace que el circuito pase al estado establecido y que $Q = 1$. Cualquier cambio en *D* mientras $CLK = 1$ no afectará la salida.

En síntesis, cuando el reloj de entrada del flip-flop disparado por borde positivo efectúa una transición positiva, el valor de *D* se transfiere a *Q*. Una transición negativa de 1 a 0 no afecta la salida, y tampoco lo hace cuando *CLK* está en el nivel estable de 1 lógico o 0 lógico. Por tanto, este tipo de flip-flop responde a la transición de 0 a 1 y a ninguna otra cosa.

Los tiempos de la respuesta de un flip-flop a los datos de entrada y al reloj se deben tomar en consideración al usar flip-flop disparados por borde. Hay un tiempo mínimo, llamado *tiempo de preparación* (*setup* en inglés), durante el cual la entrada *D* se debe mantener en un valor constante antes de que se presente la transición de reloj. Asimismo, hay un tiempo mínimo, llamado *tiempo de retención* (*hold* en inglés), durante el cual la entrada *D* no debe cambiar después de la aplicación de la transición positiva del reloj. El retardo de propagación del flip-flop se define como el intervalo de tiempo entre el borde disparador y la estabilización de la salida en un nuevo estado. Éstos y otros parámetros se especifican en los libros de datos de los fabricantes de familias lógicas específicas.

El símbolo gráfico para el flip-flop *D* disparado por flanco aparece en la figura 5-11. Es similar al utilizado para el latch *D*, excepto por el símbolo triangular antes de la letra *C* que designa una entrada *dinámica*. El *indicador dinámico* denota el hecho de que el flip-flop responde a la transición de borde del reloj. Una burbuja afuera del rectángulo, junto al indicador dinámico, denota un borde negativo para disparar el circuito. La ausencia de la burbuja denota una respuesta al borde positivo.

Otros flip-flops

Los circuitos de integración a muy grande escala contienen miles de compuertas en un paquete. Los circuitos se construyen interconectando las diversas compuertas para crear un sistema digital. Cada flip-flop se construye interconectando compuertas. El flip-flop más económico y eficiente construido de esta manera es el flip-flop *D* disparado por borde, porque es el que

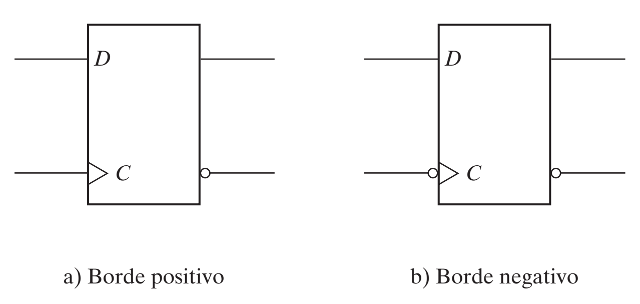
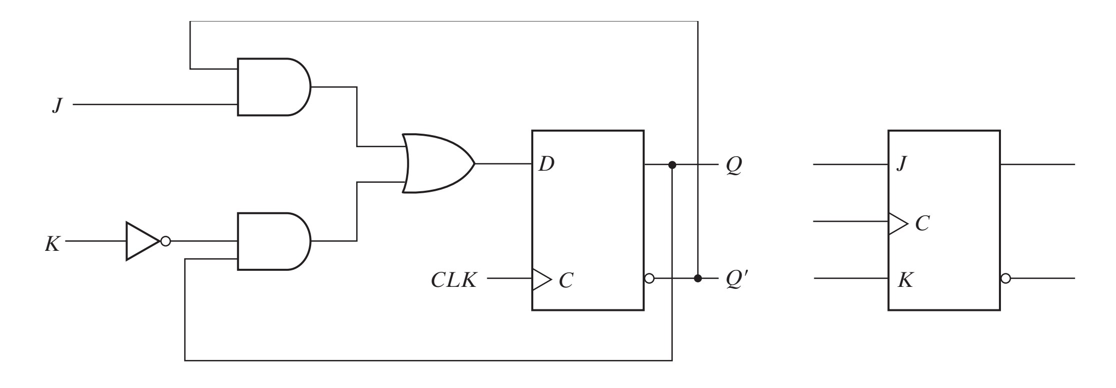


FIGURA 5-11
Símbolo gráfico para el flip-flop *D* disparado por borde



a) Diagrama de circuito b) Símbolo gráfico

FIGURA 5-12
Flip-Flop *JK*

menos compuertas requiere. Es posible construir otros tipos de flip-flops utilizando el flip-flop *D* y lógica externa. Dos flip-flops ampliamente utilizados en el diseño de sistemas digitales son los flip-flops *JK* y *T*.

Hay tres operaciones que pueden efectuarse con un flip-flop: establecerlo en 1, restablecerlo a 0 y complementar su salida. El flip-flop *JK* realiza las tres operaciones. El diagrama de circuito de un flip-flop *JK* construido con un flip-flop *D* y compuertas se reproduce en la figura 5-12a). La entrada *J* establece el flip-flop en 1, la entrada *K* lo restablece a 0 y, cuando ambas entradas están habilitadas, la salida se complementa. Esto se verifica investigando el circuito aplicado a la entrada *D:*

$$D = JQ' + K'Q$$

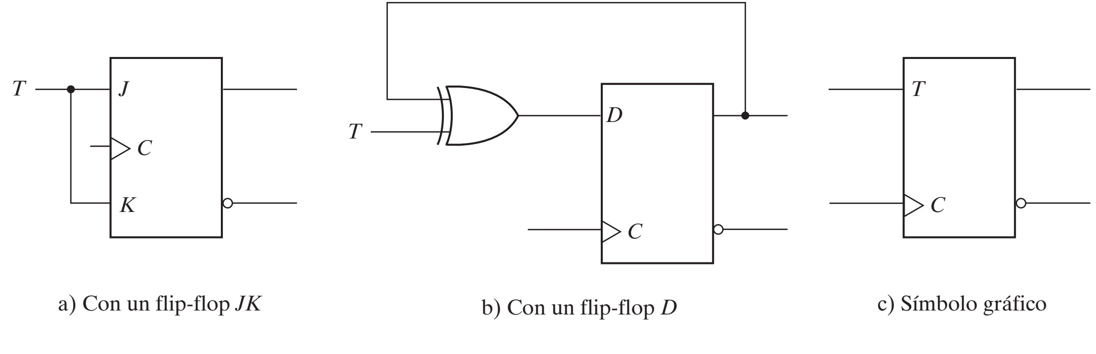
Cuando $J = 1$ y $K = 0$, $D = Q' + Q = 1$, así que el siguiente borde del reloj establece la salida en 1. Cuando $J = 0$ y $K = 1$, $D = 0$, así que el siguiente borde del reloj restablece la salida a 0. Cuando $J = 1$ y $K = 1$, $D = Q'$, y el siguiente borde del reloj complementa la salida. Cuando $J = K = 0$, $D = Q$, y el borde de reloj no altera la salida. El símbolo gráfico del flip-flop *JK* se indica en la figura 5-12b). Es similar al del flip-flop *D*, excepto que ahora las entradas están marcadas con *J* y *K*.

El flip-flop *T* (*toggle*) es un flip-flop complementador y se puede implementar con un flip-flop *JK* si se conectan entre sí las entradas *J* y *K*, como se observa en la figura 5-13a). Cuando $T = 0$ ($J = K = 0$), un borde de reloj no modifica la salida. Cuando $T = 1$ ($J = K = 1$), un borde de reloj complementa la salida. El flip-flop complementador es útil para diseñar contadores binarios.

El flip-flop *T* se puede construir con un flip-flop *D* y una compuerta OR exclusivo, como se indica en la figura 5-13b). La expresión para la entrada *D* es

$$D = T \oplus Q = TQ' + T'Q$$

Cuando $T = 0$, entonces $D = Q$, y la salida no cambia. Cuando $T = 1$, entonces $D = Q'$ y la salida se complementa. El símbolo gráfico de este flip-flop tiene una *T* en la entrada.



a) Con un flip-flop *JK* b) Con un flip-flop *D* c) Símbolo gráfico

FIGURA 5-13
Flip-Flop *T*

Tablas características

Una tabla característica define las propiedades lógicas de un flip-flop describiendo su funcionamiento en forma tabular. En la tabla 5-1 se presentan las tablas características de tres tipos de flip-flops. Definen el siguiente estado en función de las entradas y del estado actual. $Q(t)$ se refiere al estado actual antes de la aplicación de un borde de reloj. $Q(t + 1)$ es el siguiente estado, un periodo de reloj después. Observe que la entrada de borde de reloj no se incluye en la tabla característica, pero se supone implícitamente entre el tiempo t y el tiempo $t + 1$.

La tabla característica del flip-flop *JK* revela que el siguiente estado es igual al estado actual cuando las entradas J y K son ambas 0. Esto se expresa como $Q(t + 1) = Q(t)$, e indica que el reloj no produce ningún cambio de estado. Cuando $K = 1$ y $J = 0$, el reloj restablece el flip-flop y $Q(t + 1) = 0$. Con $J = 1$ y $K = 0$, el flip-flop se establece y $Q(t + 1) = 1$.

Tabla 5-1
Tablas características de flip-flops

Flip-Flop *JK*

J	K	$Q(t+1)$	
0	0	$Q(t)$	Sin cambio
0	1	0	Restablecer
1	0	1	Establecer
1	1	$Q'(t)$	Complementar

Flip-Flop *D*

D	$Q(t+1)$	
0	0	Restablecer
1	1	Establecer

Flip-Flop *T*

T	$Q(t+1)$	
0	$Q(t)$	Sin cambio
1	$Q'(t)$	Complementar

Cuando tanto J como K son 1, el siguiente estado cambia al complemento del estado actual, lo que se expresa como $Q(t + 1) = Q'(t)$.

El siguiente estado de un flip-flop D depende únicamente de la entrada D y es independiente del estado actual. Esto se expresa como $Q(t + 1) = D$, y significa que el valor del siguiente estado será igual al valor de D. Cabe señalar que el flip-flop D no tiene una condición de "sin cambio". Esta condición se obtiene inhabilitando el reloj o dejando el reloj y conectando la salida de vuelta a la entrada D cuando el estado del flip-flop no debe cambiar.

La tabla característica del flip-flop T tiene sólo dos condiciones. Cuando $T = 0$, el borde de reloj no cambia el estado. Cuando $T = 1$, el borde de reloj complementa el estado del flip-flop.

Ecuaciones características

Las propiedades lógicas de un flip-flop descritas en la tabla característica también se pueden expresar algebraicamente con una ecuación característica. En el caso del flip-flop D, tenemos la ecuación característica

$$Q(t + 1) = D$$

Esto nos dice que el siguiente estado de la salida será igual al valor de la entrada D en el estado actual. La ecuación característica para el flip-flop JK se deduce de la tabla característica o del circuito de la figura 5-12. Se obtiene

$$Q(t + 1) = JQ' + K'Q$$

donde Q es el valor de la salida del flip-flop antes de la aplicación de un borde de reloj. La ecuación característica del flip-flop T se obtiene del circuito de la figura 5-13:

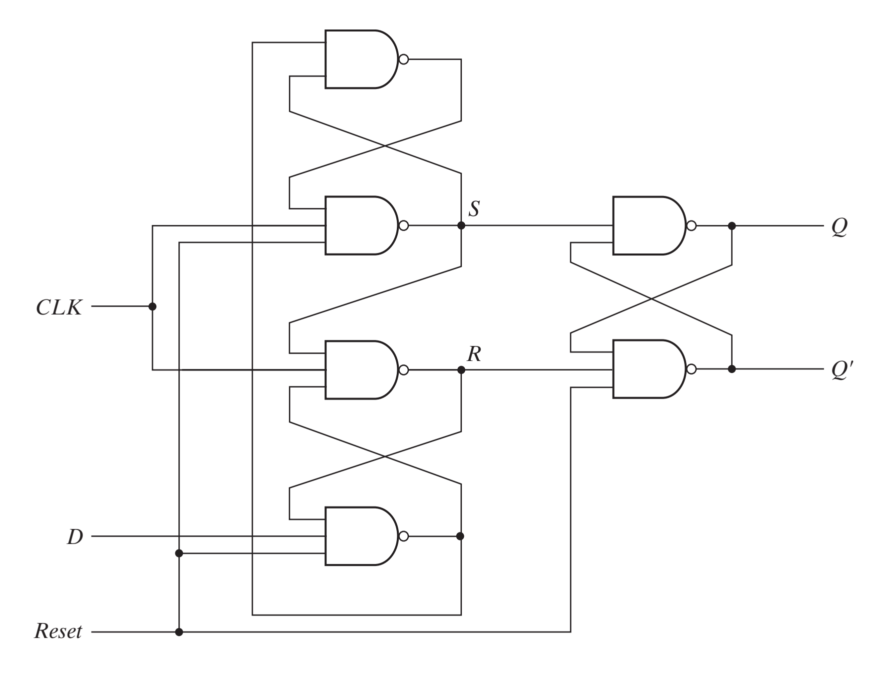
$$Q(t + 1) = T \oplus Q = TQ' + T'Q$$

Entradas directas

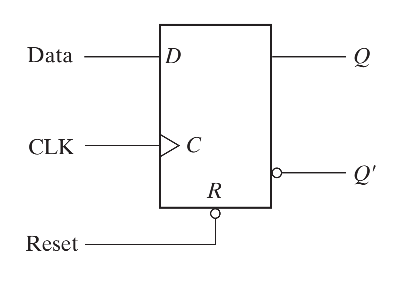
Algunos flip-flop tienen entradas asincrónicas que sirven para forzar el flip-flop a un estado dado independientemente del reloj. La entrada que pone el flip-flop en 1 se llama *preestablecimiento* (*preset*) o *establecimiento directo*. La entrada que pone en 0 el flip-flop se llama *borrado* (*clear*) o *restablecimiento directo*. Cuando se enciende un sistema digital, se desconoce el estado de los flip-flops. Las entradas directas sirven para poner todos los flip-flops del sistema en un estado inicial conocido antes del funcionamiento con reloj.

En la figura 5-14 se ilustra un flip-flop D disparado por flanco positivo, con restablecimiento asincrónico. El diagrama de circuito es igual al de la figura 5-10, excepto por la entrada de restablecimiento (*reset*) adicional conectada a tres compuertas NAND. Cuando la entrada de restablecimiento es 0, hace que Q' se mantenga en 1; esto, a su vez, pone en 0 la salida Q, con lo que el flip-flop se restablece. Otras dos conexiones de la entrada de restablecimiento garantizan que la entrada S del tercer latch SR se mantenga en 1 lógico mientras la entrada de restablecimiento está en 0, sean cuales sean los valores de D y de CLK.

El símbolo gráfico del flip-flop D con restablecimiento directo tiene una entrada adicional que se marca con R. La burbuja en la entrada indica que el restablecimiento está activo en el nivel de 0 lógico. Los flip-flops con establecimiento directo utilizan el símbolo S para la entrada de establecimiento asincrónico.



a) Diagrama de circuito



b) Símbolo gráfico

R	C	D	Q	Q'
0	X	X	0	1
1	↑	0	0	1
1	↑	1	1	0

c) Tabla de función

FIGURA 5-14
Flip-flop *D* con restablecimiento asincrónico

La tabla de función especifica la operación del circuito. Cuando $R = 0$, la salida se restablece a 0. Este estado es independiente de los valores de D o de C. El funcionamiento normal con reloj sólo podrá iniciarse después de que la entrada de restablecimiento cambie a 1 lógico. El reloj en C lleva una flecha hacia arriba para indicar que el flip-flop se dispara con el borde positivo del reloj. El valor en D se transfiere a Q con cada señal de reloj de borde positivo, siempre que $R = 1$.

5-4 ANÁLISIS DE CIRCUITOS SECUENCIALES CON RELOJ

El comportamiento de un circuito secuencial con reloj está determinado por las entradas, las salidas y el estado de sus flip-flops. Las salidas y el siguiente estado son función de las entradas y del estado actual. El análisis de un circuito secuencial consiste en obtener una tabla o diagrama para la sucesión temporal de entradas, salidas y estados internos. También es posible escribir expresiones booleanas que describan el comportamiento del circuito secuencial. Tales expresiones deberán incluir la sucesión temporal necesaria, sea directa o indirectamente.

Un diagrama lógico se reconoce como circuito secuencial con reloj si incluye flip-flops con entradas de reloj. Los flip-flops pueden ser de cualquier tipo, y el diagrama lógico podría incluir o no compuertas de circuitos combinacionales. En esta sección se incluye una representación algebraica para especificar la condición del siguiente estado en términos del estado actual y de las entradas. Luego se presentará una tabla de estados y un diagrama de estados para describir el comportamiento del circuito secuencial. Mostraremos otra representación algebraica para especificar el diagrama lógico de los circuitos secuenciales. Se incluyen también ejemplos específicos para ilustrar los diversos procedimientos.

Ecuaciones de estado

El comportamiento de los circuitos secuenciales con reloj se describe algebraicamente con ecuaciones de estado. Una *ecuación de estado* (también llamada *ecuación de transición*) especifica el siguiente estado en función del estado actual y las entradas. Consideremos el circuito secuencial de la figura 5-15. Consta de dos flip-flops D, A y B, una entrada x y una salida y. Puesto que la entrada D de un flip-flop determina el valor del siguiente estado, podemos escribir un conjunto de ecuaciones de estado para el circuito:

$$A(t + 1) = A(t)x(t) + B(t)x(t)$$
$$B(t + 1) = A'(t)x(t)$$

Una ecuación de estado es una expresión algebraica que especifica la condición para una transición de estado de un flip-flop. El miembro izquierdo de la ecuación, donde aparece $(t + 1)$, denota el siguiente estado del flip-flop, un borde de reloj después. El miembro derecho de la ecuación es una expresión booleana que especifica el estado actual y las condiciones de entrada que harán que el siguiente estado sea 1. Puesto que todas las variables de las expresiones booleanas son función del estado actual, se omite la designación (t) después de cada variable, por conveniencia, a fin de expresar las ecuaciones de estado en la forma más compacta:

$$A(t + 1) = Ax + Bx$$
$$B(t + 1) = A'x$$

Las expresiones booleanas para las ecuaciones de estado se deducen directamente de las compuertas que forman la parte de circuito combinacional del circuito secuencial, ya que los valores D del circuito combinacional determinan el siguiente estado. Asimismo, el valor del estado actual de la salida se expresa algebraicamente como

$$y(t) = [A(t) + B(t)]x'(t)$$

Al omitir el símbolo (t) para el estado actual, se obtiene la ecuación booleana de salida:

$$y = (A + B)x'$$

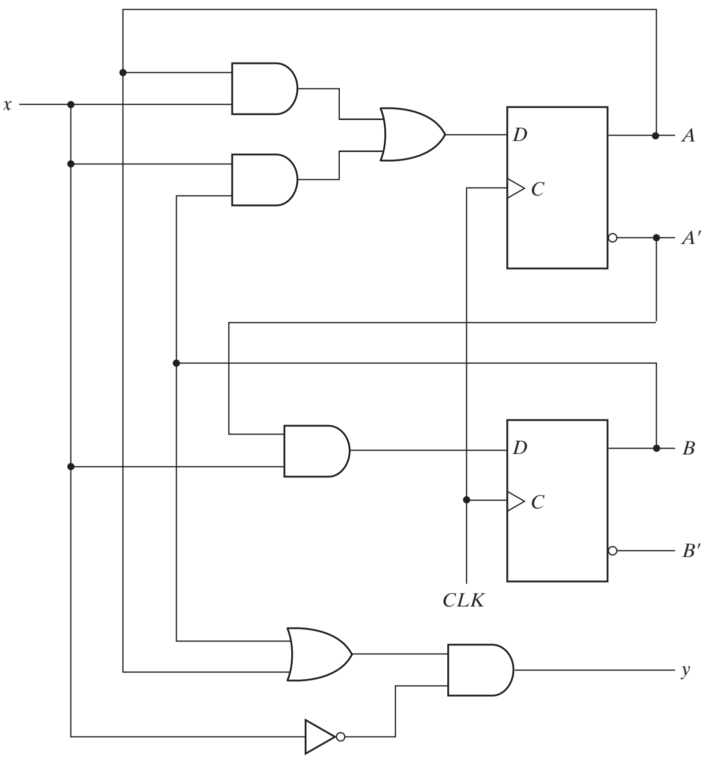


FIGURA 5-15
Ejemplo de circuito secuencial

Tabla de estados

La sucesión temporal de entradas, salidas y estados de flip-flop se puede presentar de forma compacta en una *tabla de estados* (también llamada *tabla de transición*). La tabla de estados para el circuito de la figura 5-15 se presenta en la tabla 5-2. La tabla consta de cuatro secciones rotuladas *estado actual*, *entrada*, *siguiente estado* y *salida*. La sección de estado actual muestra los estados de los flip-flops A y B en cualquier instante dado t. La sección de entrada da un valor de x para cada posible estado actual. La sección de siguiente estado muestra los estados de los flip-flops un ciclo de reloj después, en el tiempo $t + 1$. La sección de salida da el valor de y en el tiempo t para cada estado actual y condición de entrada.

La preparación de una tabla de estados requiere enumerar todas las posibles combinaciones binarias de estado actual y entradas. En este caso, tenemos ocho combinaciones binarias, de 000 a 111. Luego se determinan los valores de siguiente estado a partir del diagrama lógico o de las ecuaciones de estado. El siguiente estado del flip-flop A deberá satisfacer la ecuación de estado

$$A(t + 1) = Ax + Bx$$

La sección de siguiente estado en la columna A de la tabla de estados tiene tres unos donde el estado actual y el valor de entrada satisfacen las condiciones de que el estado actual de A y la

Tabla 5-2
Tabla de estados para el circuito de la figura 5-15

Estado actual		Entrada	Siguiente estado		Salida
A	*B*	*x*	*A*	*B*	*y*
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

entrada x son ambos 1, o el estado actual de B y la entrada x son ambos 1. De forma similar, el siguiente estado del flip-flop B se deduce de la ecuación de estado

$$B(t + 1) = A'x$$

y es igual a 1 cuando el estado actual de A es 0 y la entrada x es 1. La columna de salida se deduce de la ecuación de salida

$$y = Ax' + Bx'$$

La tabla de estados de un circuito secuencial con flip-flops tipo D se obtiene por el mismo procedimiento delineado en el ejemplo anterior. En general, un circuito secuencial con m flip-flops y n entradas necesita 2^{m+n} filas en la tabla de estados. Se hace una lista de los números binarios del 0 hasta $2^{m+n} - 1$ bajo las columnas de estado actual y entrada. La sección de siguiente estado tiene m columnas, una para cada flip-flop. Los valores binarios para el siguiente estado se deducen directamente de las ecuaciones de estado. La sección de salida tiene tantas columnas como variables de salida haya. Su valor binario se deduce del circuito o de la función booleana de la misma manera que se deduce una tabla de verdad.

A veces es conveniente expresar la tabla de estados en una forma un poco distinta. En la otra configuración, la tabla de estados sólo tiene tres secciones: estado actual, siguiente estado y salida. Las condiciones de entrada se enumeran en las secciones de siguiente estado y salida. En la tabla 5-3 se repite la tabla de estados de la tabla 5-2, en el segundo formato. Para cada estado actual, hay dos siguientes estados y salidas posibles, dependiendo del valor de la entrada. Una forma podría ser preferible a la otra, dependiendo de la aplicación.

Tabla 5-3
Segunda forma de la tabla de estados

Estado actual	Siguiente estado		Salida	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
AB	*AB*	*AB*	*y*	*y*
00	00	01	0	0
01	00	11	1	0
10	00	10	1	0
11	00	10	1	0

Diagrama de estados

La información contenida en una tabla de estados se representa gráficamente en forma de diagrama de estados. En este tipo de diagramas, un estado se representa con un círculo, y las transiciones entre estados se indican con flechas que conectan a los círculos. En la figura 5-16 se aprecia el diagrama de estados del circuito secuencial de la figura 5-15. El diagrama de estados proporciona la misma información que la tabla de estados y se obtiene directamente de la tabla 5-2 o 5-3. El número binario dentro de cada círculo identifica el estado de los flip-flops. Las flechas se rotulan con dos números binarios separados por una diagonal. Primero se da el valor de entrada durante el estado actual, y el número después de la diagonal indica la salida durante el estado actual, con esa entrada. (Es importante recordar que el valor de bit indicado para la salida a lo largo de la flecha se da durante el estado actual y con la entrada indicada, y nada tiene que ver con la transición al siguiente estado.) Por ejemplo, la flecha del estado 00 a 01 lleva el rótulo 1/0, lo que significa que cuando el circuito secuencial está en el estado actual 00 y la entrada es 1, la salida es 0. Después del siguiente ciclo de reloj, el circuito pasa al siguiente estado, 01. Si la entrada cambia a 0, la salida será 1, pero si la entrada sigue siendo 1, la salida se mantendrá en 0. Esta información se obtiene del diagrama de estados siguiendo las dos flechas que salen del círculo correspondiente al estado 01. Una flecha que conecta a un círculo consigo mismo indica que no hay cambio de estado.

No hay diferencia entre una tabla de estados y un diagrama de estados, como no sea en la forma de representación. La tabla de estados se deduce más fácilmente de un diagrama lógico dado y la ecuación de estado. El diagrama de estados se sigue directamente de la tabla de estados. El diagrama de estados muestra una perspectiva gráfica de las transiciones de estado y es la forma más apropiada para interpretar el funcionamiento del circuito, si quien lo interpreta es un ser humano. Por ejemplo, el diagrama de estados de la figura 5-16 indica claramente que, partiendo del estado 00, la salida será 0 en tanto la entrada se mantenga en 1. La primera entrada 0 después de una serie de unos da una salida de 1 y transfiere al circuito de vuelta al estado inicial 00.

Ecuaciones de entrada de flip-flops

El diagrama lógico de un circuito secuencial consiste en flip-flops y compuertas. Las interconexiones de compuertas forman un circuito combinacional y podrían especificarse algebraicamente con expresiones booleanas. El conocimiento del tipo de flip-flops y una lista de las expresiones booleanas del circuito combinacional proporcionan la información necesaria para dibujar el dia-

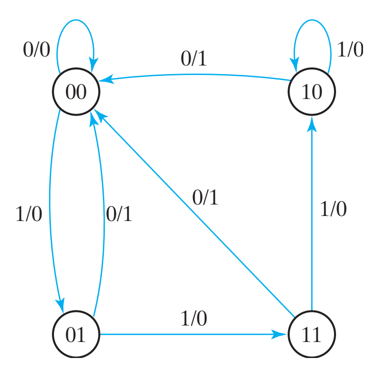


FIGURA 5-16
Diagrama de estados del circuito de la figura 5-15

grama lógico del circuito secuencial. La parte del circuito combinacional que genera salidas externas se describe algebraicamente con un conjunto de funciones booleanas llamadas *ecuaciones de salida*. La parte del circuito que genera las entradas a los flip-flops se describe algebraicamente con un conjunto de funciones booleanas llamadas *ecuaciones de entrada* de flip-flops (o *ecuaciones de excitación*). Adoptaremos la convención de usar el símbolo de entrada de flip-flop para denotar la variable de ecuación de entrada y un subíndice para indicar el nombre de la salida de flip-flop. Por ejemplo, la ecuación de entrada siguiente especifica la compuerta OR con entradas x y y conectada a la entrada D de un flip-flop cuya salida se rotula con el símbolo Q:

$$D_Q = x + y$$

El circuito secuencial de la figura 5-15 consta de dos flip-flops D, A y B, una entrada x y una salida y. El diagrama lógico del circuito se expresa algebraicamente con dos ecuaciones de entrada de flip-flops y una ecuación de salida:

$$D_A = Ax + Bx$$
$$D_B = A'x$$
$$y = (A + B)x'$$

Las tres ecuaciones proporcionan la información necesaria para dibujar el diagrama lógico del circuito secuencial. El símbolo D_A especifica un flip-flop D rotulado A. D_B especifica un segundo flip-flop D rotulado B. Las expresiones booleanas asociadas a estas dos variables, y la expresión de la salida y, especifican la parte de circuito combinacional del circuito secuencial.

Las ecuaciones de entrada de flip-flop son una forma algebraica conveniente para especificar el diagrama lógico de un circuito secuencial. Implican el tipo de flip-flop con el símbolo de letra, y especifican cabalmente el circuito combinacional que alimenta a los flip-flops. Cabe señalar que la expresión de la ecuación de entrada de un flip-flop D es idéntica a la expresión de la ecuación de estado correspondiente. Ello se debe a la ecuación característica que iguala el siguiente estado al valor de la entrada D: $Q(t + 1) = D_Q$.

Análisis con flip-flops *D*

Resumiremos el procedimiento para analizar un circuito secuencial con reloj con flip-flops D utilizando un ejemplo sencillo. El circuito que queremos analizar se describe con la ecuación de entrada

$$D_A = A \oplus x \oplus y$$

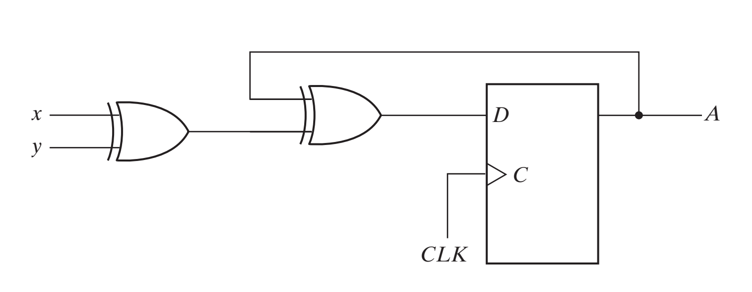
El símbolo D_A implica un flip-flop D con salida A. Las variables x y y son las entradas del circuito. No se dan ecuaciones de salida, así que la salida proviene implícitamente de la salida del flip-flop. Obtenemos el diagrama lógico de la ecuación de entrada [figura 5-17a)].

La tabla de estados tiene una columna para el estado actual del flip-flop A, dos columnas para las dos entradas y una columna para el siguiente estado de A. Los números binarios bajo Axy van de 000 a 111, como se observa en la figura 5-17b). Los valores de siguiente estado se obtienen de la ecuación de estado

$$A(t + 1) = A \oplus x \oplus y$$

La expresión especifica una función impar y es igual a 1 cuando sólo una variable es 1 o cuando las tres variables son 1. Esto se indica en la columna de siguiente estado de A.

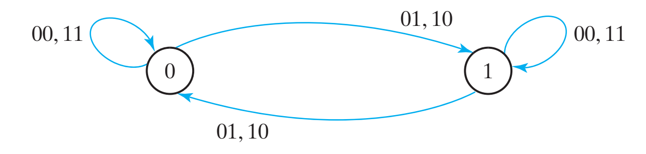
El circuito tiene un flip-flop y dos estados. El diagrama de estados consiste en dos círculos, uno para cada estado [figura 5-17c)]. El estado actual y la salida pueden ser 0 o 1, como indica el número dentro de los círculos. No se necesita una diagonal en las flechas porque no hay salida de circuito combinacional. Las dos entradas pueden tener cuatro posibles combinacio-



a) Diagrama de circuito

Estado actual	Salidas		Siguiente estado
A	x	y	A
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

b) Tabla de estados



c) Diagrama de estados

FIGURA 5-17
Circuito secuencial con flip-flop *D*

nes para cada estado. Las dos combinaciones de entrada durante cada transición de estado se separan con una coma para simplificar la notación.

Análisis con flip-flops *JK*

Una tabla de estados consta de cuatro secciones: estado actual, entradas, siguiente estado y salidas. Las dos primeras se obtienen enumerando todas las combinaciones binarias. La sección de salida se determina con base en las ecuaciones de salida. Los valores de siguiente estado se obtienen de las ecuaciones de estado. En el caso de un flip-flop tipo D, la ecuación de estado es igual a la ecuación de entrada. Cuando se usa un flip-flop de otro tipo, como un JK o un T, es necesario consultar la tabla característica o ecuación característica correspondiente para obtener los valores de siguiente estado. Ilustraremos el procedimiento primero utilizando la tabla característica, y luego lo repetiremos usando la ecuación característica.

Los valores de siguiente estado de un circuito secuencial que usa flip-flops tipo JK o T se deducen con el procedimiento siguiente:

1. Determine las ecuaciones de entrada del flip-flop en términos del estado actual y las variables de entrada.
2. Enumere los valores binarios de cada ecuación de entrada.
3. Use la tabla característica del flip-flop en cuestión para determinar los valores de siguiente estado de la tabla de estados.

Por ejemplo, consideremos el circuito secuencial con dos flip-flops JK, A y B, y una entrada, x, que se ilustra en la figura 5-18. El circuito no tiene salidas, de modo que la tabla de estados no necesita una columna de salida. (Las salidas de los flip-flops se consideran como las salidas en este caso.)

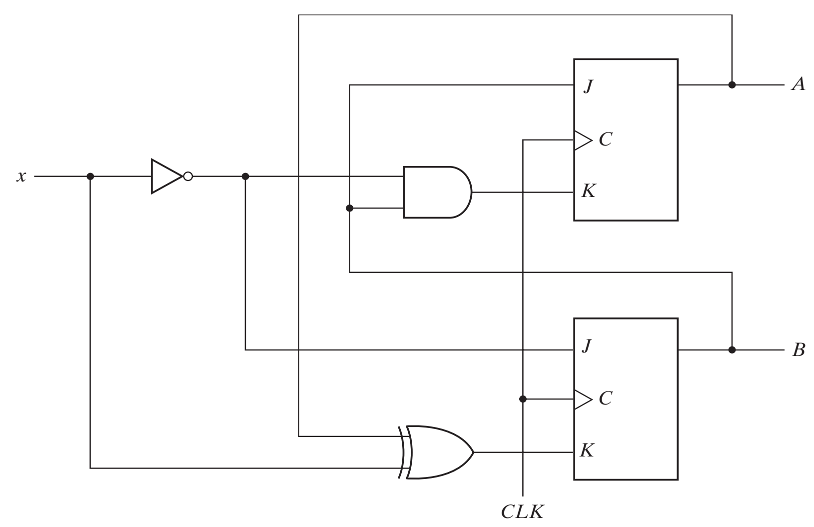


FIGURA 5-18
Circuito secuencial con flip-flop *JK*

El circuito se puede especificar con las ecuaciones de entrada de flip-flop

$$J_A = B \qquad K_A = Bx'$$
$$J_B = x' \qquad K_B = A'x + Ax' = A \oplus x$$

La tabla de estados del circuito secuencial aparece en la tabla 5-4. Las columnas de estado actual y entrada presentan las ocho combinaciones binarias. Los valores binarios de las columnas rotuladas *entradas de flip-flops* no forman parte de la tabla de estados, pero las necesitamos para evaluar el siguiente estado, como especifica el paso 2 del procedimiento. Estos valores binarios se obtienen directamente de las cuatro ecuaciones de entrada de forma similar a como se deduce una tabla de verdad de una expresión booleana. El siguiente estado de cada flip-flop se evalúa a partir de las entradas J y K correspondientes y la tabla característica del flip-flop JK dada en la tabla 5-1. Debemos considerar cuatro casos. Cuando $J = 1$ y $K = 0$, el siguiente estado es 1. Cuando $J = 0$ y $K = 1$, el siguiente estado es 0. Cuando $J = K = 0$, no hay cambio de estado y el valor de siguiente estado es igual al de estado actual. Cuando $J = K = 1$,

Tabla 5-4
Tabla de estados de un circuito secuencial con flip-flops JK

Estado Actual		Entrada	Siguiente estado		Entradas de flip-flop			
A	*B*	*x*	*A*	*B*	J_A	K_A	J_B	K_B
0	0	0	0	1	0	0	1	0
0	0	1	0	0	0	0	0	1
0	1	0	1	1	1	1	1	0
0	1	1	1	0	1	0	0	1
1	0	0	1	1	0	0	1	1
1	0	1	1	0	0	0	0	0
1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	0	0	0

el bit de siguiente estado es el complemento del bit de estado actual. Se dan ejemplos de los últimos dos casos de la tabla cuando el estado actual *AB* es 10 y la entrada *x* es 0. *JA* y *KA* son ambos 0 y el estado actual de *A* es 1. Por tanto, el siguiente estado de *A* sigue siendo el mismo y es igual a 1. En la misma fila de la tabla, *JB* y *KB* son ambos 1. Puesto que el estado actual de *B* es 0, el siguiente estado de *B* se complementará y cambiará a 1.

También es posible obtener los valores de siguiente estado evaluando las ecuaciones de estado de la ecuación característica, mediante el procedimiento siguiente:

1. Obtenga las ecuaciones de entrada de flip-flop en términos del estado actual y las variables de entrada.
2. Sustituya las ecuaciones de salida en la ecuación característica del flip-flop para obtener las ecuaciones de estado.
3. Use las ecuaciones de estado correspondientes para determinar los valores de siguiente estado de la tabla de estados.

Las ecuaciones de entrada para los dos flip-flops *JK* de la figura 5-18 se presentaron en la página anterior. Obtenemos las ecuaciones características de los flip-flops sustituyendo *A* o *B* por el nombre del flip-flop, en vez de *Q*:

$$A(t+1) = JA' + K'A$$
$$B(t+1) = JB' + K'B$$

Sustituyendo los valores de J_A y K_A de las ecuaciones de entrada, se obtiene la ecuación de estado para *A*:

$$A(t+1) = BA' + (Bx')'A = A'B + AB' + Ax$$

La ecuación de estado proporciona los valores de bits para la columna de siguiente estado de *A* en la tabla de estados. De forma similar, la ecuación de estado del flip-flop *B* se deduce de la ecuación característica sustituyendo los valores de J_B y K_B:

$$B(t+1) = x'B' + (A \oplus x)'B = B'x' + ABx + A'Bx'$$

La ecuación de estado proporciona los valores de bit para la columna de siguiente estado de *B* en la tabla de estados. Observe que las columnas de entradas de flip-flop de la tabla 5-4 no se necesitan cuando se usan ecuaciones de estado.

El diagrama de estados del circuito secuencial se presenta en la figura 5-19. Vemos que, como el circuito no tiene salidas, las flechas que salen de los círculos se marcan con un solo número binario para indicar el valor de la entrada *x*.

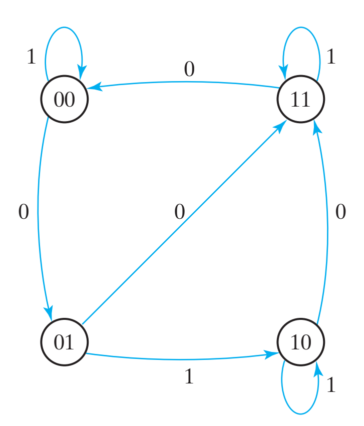


FIGURA 5-19
Diagrama de estados del circuito de la figura 5-18

Análisis con flip-flops *T*

El análisis de un circuito secuencial con flip-flops *T* sigue el mismo procedimiento que delineamos para los flip-flops *JK*. Los valores de siguiente estado de la tabla de estados se obtienen utilizando la tabla característica de la tabla 5-1 o bien la ecuación característica

$$Q(t+1) = T \oplus Q = T'Q + TQ'$$

Consideremos el circuito secuencial de la figura 5-20. Tiene dos flip-flops *A* y *B*, una entrada *x* y una salida *y*. Se describe algebraicamente con dos ecuaciones de entrada y una de salida:

$$T_A = Bx$$
$$T_B = x$$
$$y = AB$$

La tabla de estados del circuito se presenta en la tabla 5-5. Los valores de *y* se obtienen de la ecuación de salida. Los valores para el siguiente estado se deducen de las ecuaciones de estado sustituyendo T_A y T_B en las ecuaciones características para dar

$$A(t+1) = (Bx)'A + (Bx)A' = AB' + Ax' + A'Bx$$
$$B(t+1) = x \oplus B$$

Los valores de siguiente estado para *A* y *B* en la tabla de estados se obtienen de las expresiones para las dos ecuaciones de estado.

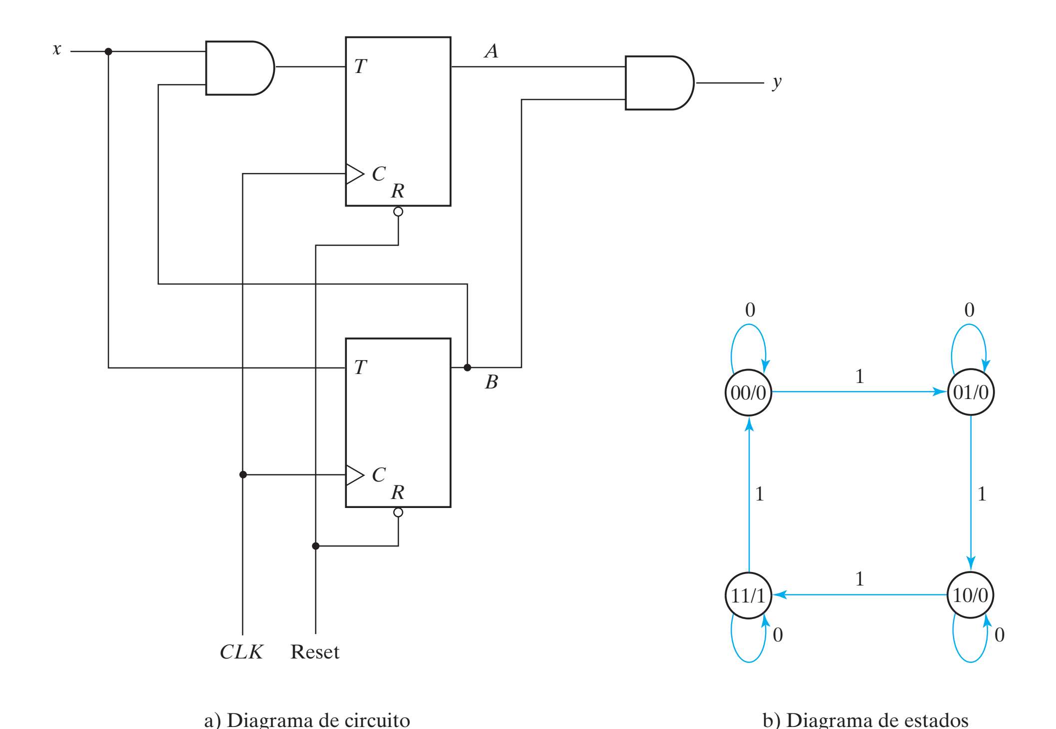


FIGURA 5-20
Circuito secuencial con dos flip-flops *T*

Tabla 5-5
Tabla de estados para un circuito secuencial con flip-flops T

Estado actual		Entrada	Siguiente estado		Salida
A	*B*	*x*	*A*	*B*	*y*
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	1

El diagrama de estados del circuito se reproduce en la figura 5-20b). En tanto la entrada x sea 1, el circuito se comportará como un contador binario con la sucesión de estados 00, 01, 10, 11 y de vuelta a 00. Cuando $x = 0$, el circuito permanece en el mismo estado. La salida y es 1 cuando el estado actual es 11. Aquí la salida depende únicamente del estado actual y es independiente de la entrada. Los dos valores separados por una diagonal dentro de cada círculo corresponden al estado actual y a la salida.

Modelos Mealy y Moore

El modelo más general de un circuito secuencial tiene entradas, salidas y estados internos. Se acostumbra distinguir entre dos modelos de circuitos secuenciales: el modelo Mealy y el modelo Moore. Difieren en la forma en que se genera la salida. En el modelo Mealy, la salida es función tanto del estado actual como de la entrada. En el modelo Moore, la salida sólo es función del estado actual. Al tratar los dos modelos, algunos libros y otras fuentes técnicas ven el circuito secuencial como una máquina de estados finitos (FSM, *finite state machine*). El modelo Mealy de un circuito secuencial es una FSM Mealy o máquina Mealy. El modelo Moore es una FSM Moore o máquina Moore.

En la figura 5-15 se ilustra un ejemplo de modelo Mealy. La salida y es función tanto de la entrada x como del estado actual de A y B. El diagrama de estados correspondiente de la figura 5-16 muestra los valores de entrada y de salida separados por una diagonal sobre las flechas entre los estados.

En la figura 5-18 aparece un ejemplo de modelo Moore. Aquí la salida es función únicamente del estado actual. El diagrama de estados correspondiente de la figura 5-19 sólo tiene entradas marcadas sobre las flechas. Las salidas son los estados de flip-flop indicados dentro de los círculos. Otro ejemplo de modelo Moore es el circuito secuencial de la figura 5-20. La salida depende únicamente de los valores de los flip-flops, así que sólo es función del estado actual. El valor de la entrada se marca en el diagrama de estados junto a las flechas, mientras que el valor de salida se indica dentro del círculo junto con el estado actual.

En un modelo Moore, las salidas del circuito secuencial se sincronizan con el reloj porque sólo dependen de salidas de flip-flop que están sincronizadas con el reloj. En un modelo Mealy, las salidas podrían cambiar si las entradas cambian durante el ciclo de reloj. Además, las sali-

das podrían tener valores falsos momentáneos debidos al retardo entre el momento en que las entradas cambian y el momento en que cambian las salidas de flip-flop. Para sincronizar un circuito tipo Mealy, las entradas del circuito secuencial se deben sincronizar con el reloj y las salidas se deben muestrear únicamente durante el borde del reloj.

5-5 HDL PARA CIRCUITOS SECUENCIALES

Presentamos el lenguage de descripción de hardware (HDL) Verilog en la sección 3-9. En la sección 4-11 se hizo una descripción de los circuitos combinacionales y una introducción al modelado de comportamiento. En esta sección seguiremos estudiando el modelado de comportamiento y presentaremos ejemplos de descripciones de flip-flops y circuitos secuenciales.

Modelado de comportamiento

Hay dos tipos de enunciados de comportamiento en Verilog HDL: *inicial* y *siempre*. El comportamiento inicial se ejecuta una vez en el tiempo = 0. El comportamiento "siempre" se ejecuta una y otra vez hasta que la simulación termina. Los comportamientos se declaran dentro de los módulos con las palabras clave **initial** y **always** seguidas de un enunciado o bloque de enunciados delimitado por las palabras clave **begin** y **end.** Un módulo puede contener un número arbitrario de enunciados **initial** o **always.** Estos enunciados se ejecutan de forma concurrente a partir del tiempo 0.

Un enunciado **initial** se ejecuta una sola vez. Inicia su ejecución al principio de la simulación y termina una vez que han terminado de ejecutarse todos los enunciados. Como se mencionó al final de la sección 4-11, el enunciado **initial** es útil para generar señales de entrada a fin de simular un diseño. Al simular un circuito secuencial, es necesario generar una fuente de reloj para disparar los flip-flops. He aquí dos posibles formas de incluir un reloj de operación libre:

```
initial
   begin
      clock = 1'b0 ;
      repeat (30)
      #10 clock = ~ clock;
   end
```

```
initial
    begin
      clock = 1'b0;
      #300  $finish;
    end
always
      #10  clock = ~clock;
```

En la primera versión, el bloque **initial** está encerrado entre las palabras clave **begin** y **end.** El reloj se pone en 0 en el tiempo = 0; se complementa cada 10 unidades de tiempo y se repite 30 veces. Esto produce 15 ciclos de reloj, cada uno con una duración de 20 unidades de tiempo. En la segunda versión, el bloque **initial** pone el reloj en 0 en el tiempo = 0. Después de 10 unidades de tiempo, el enunciado **always** complementa repetidamente el reloj cada 10 unidades de tiempo, lo que proporciona un reloj con una duración de 20 unidades de tiempo. La simulación termina en respuesta a la tarea del sistema **$finish** en el tiempo = 300.

El enunciado **always** se controla con retardos que esperan cierto tiempo o esperan hasta que ciertas condiciones se cumplen o se presentan ciertos sucesos. Aquí sólo se explicará la condición de control por suceso. Este tipo de enunciado tiene la forma

```
always @ (expresión de control de sucesos)
Enunciados procedimentales de asignación.
```

La expresión de control de sucesos especifica la condición que debe presentarse para activar la ejecución de los enunciados procedimentales de asignación. Las variables del miembro iz-

quierdo de los enunciados procedimentales deben ser del tipo de datos **reg** y declararse como tales. El miembro derecho puede ser cualquier expresión que produzca un valor empleando operadores definidos en Verilog.

La expresión de control de sucesos (también llamada lista de sensibilidad) especifica los sucesos que deben darse para iniciar la ejecución de los enunciados procedimentales del bloque **always.** Los enunciados de ese bloque se ejecutan sucesivamente y la ejecución se suspende después del último enunciado. Luego, el enunciado **always** espera otra vez que se presente un suceso. Aquí consideraremos dos tipos de sucesos: sucesos sensibles al nivel y sucesos disparados por flanco. Los primeros se dan en los circuitos combinacionales y en latches. Por ejemplo, el enunciado

```
always @ (A or B or Reset)
```

hace que se ejecuten los enunciados procedimentales del bloque **always** si hay un cambio en *A* o en *B* o en *Reset*. En los circuitos secuenciales sincrónicos, los flip-flops sólo deben cambiar como respuesta a una transición de pulso de reloj. La transición podría ser un disparador de borde positivo o de borde negativo. Verilog HDL maneja estas condiciones con dos palabras clave: **posedge** y **negedge.** Por ejemplo,

```
always @(posedge clock or negedge reset)
```

hace que se ejecuten los enunciados procedimentales sólo si el reloj sufre una transición positiva o si *Reset* pasa por una transición negativa.

Los enunciados procedimentales son enunciados contenidos en un enunciado **initial** o **always.** Esto contrasta con las asignaciones continuas que vimos en la sección 4-11 al hablar del modelado de flujo de datos, donde el enunciado se evalúa continuamente. Hay dos tipos de enunciados procedimentales, *bloqueadores* y *no bloqueadores*, y se distinguen por los símbolos que usan. Los enunciados bloqueadores emplean el símbolo (=) como operador de asignación, mientras que los no bloqueadores usan el operador (<=). Los enunciados de asignación bloqueadores se ejecutan sucesivamente en el orden en que aparecen en un bloque secuencial. Los enunciados no bloqueadores evalúan las expresiones del miembro derecho pero no efectúan la asignación al miembro izquierdo sino hasta que se han evaluado todas las expresiones. Se entenderán mejor los dos tipos de asignaciones con un ejemplo. Consideremos las dos asignaciones procedimentales bloqueadoras:

```
B = A
C = B + 1
```

El primer enunciado transfiere *A* a *B*. El segundo enunciado incrementa el nuevo valor de *B* y lo transfiere a *C*. Al final, *C* contiene el valor de $A + 1$.

Consideremos ahora los dos enunciados en forma de asignaciones no bloqueadoras:

```
B <= A
C <= B + 1
```

Cuando se ejecutan los enunciados, las expresiones de la derecha se evalúan y se almacenan en un lugar temporal. El valor de *A* se guarda en un lugar y el valor de $B + 1$ se guarda en otro. Una vez que se han evaluado y almacenado todas las expresiones del bloque secuencial, se efectúa la asignación a los destinos de la izquierda. En este caso, *C* contendrá el valor original de *B* más uno. Casi todos los ejemplos de este capítulo y el siguiente pueden usar enunciados bloqueadores. Los enunciados no bloqueadores son indispensables cuando se efectúa diseño en el nivel de transferencia de registros, como se verá en el capítulo 8.

Flip-flops y latches

Los ejemplos HDL 5-1 a 5-4 muestran descripciones de diversos flip-flops y un latch *D*. El latch *D* es transparente y responde a un cambio en la entrada de datos con un cambio en la salida en tanto la entrada de control esté habilitada. El módulo de descripción del latch *D* se presenta en el ejemplo HDL 5-1. Tiene dos entradas, *D* y *control*, y una salida, *Q*. Puesto que *Q* se evalúa en un enunciado procedimental, se le debe declarar como de tipo **reg.** Los latches responden a niveles de señal de entrada, así que las dos entradas se dan sin calificadores de borde (**posedge** o **negedge**) en la expresión de control de sucesos que sigue al símbolo @ en el enunciado **always.** Sólo hay un enunciado de asignación procedimental bloqueador, y especifica la transferencia de la entrada *D* a la salida *Q* si el control es 1 lógico. Advierta que este enunciado se ejecuta cada vez que hay un cambio en *D* si el control es 1.

El ejemplo HDL 5-2 describe dos flip-flops *D* de borde positivo en dos módulos. El primero responde únicamente al reloj; el segundo incluye una entrada de restablecimiento asincrónico. La salida *Q* debe declararse como de tipo de datos **reg** además de darse como salida. El motivo es que es una salida de destino en un enunciado procedimental de asignación. La palabra clave **posedge** garantiza que la transferencia de la entrada *D* a *Q* sólo se dará durante la transición de borde positivo de CLK. Un cambio en *D* en cualquier otro momento no cambia *Q*.

Ejemplo HDL 5-1

```
//Descripción de un latch D (Véase la figura 5-6)
module D_latch (Q,D,control);
   output Q;
   input D,control;
   reg Q;
   always @ (control or D)
   if (control) Q = D;     //Igual que: if (control == 1)
endmodule
```

Ejemplo HDL 5-2

```
//Flip-flop D
module D_FF (Q,D,CLK);
   output Q;
   input D,CLK;
   reg Q;
   always @ (posedge CLK)
     Q = D;
endmodule

//Flip-flop D con restablecimiento asincrónico.
module DFF (Q,D,CLK,RST);
   output Q;
   input D,CLK,RST;
   reg Q;
   always @(posedge CLK or negedge RST)
     if (~RST) Q = 1'b0;    // Igual a: if (RST == 0)
     else Q = D;
endmodule
```

El segundo módulo incluye una entrada de restablecimiento asincrónico además del reloj sincrónico. Se usa una forma especial del enunciado **if** para generar este tipo de flip-flop. La expresión de sucesos después del símbolo @ en el enunciado **always** puede tener cualquier número de sucesos de borde, sean **posedge** o **negedge.** Uno de ellos debe ser un suceso de reloj. Los demás especifican condiciones en las que debe ejecutarse lógica asincrónica. Cada enunciado **if** o **else if** de los enunciados procedimentales de asignación corresponde a un suceso asincrónico. El último enunciado **else** corresponde al suceso de reloj. Hay dos sucesos de borde en el segundo módulo del ejemplo 5-2. El suceso **negedge** RST (restablecimiento) es asincrónico porque equivale al suceso **if** (~RST). En tanto RST sea 0, Q se pondrá en 0. Si CLK tiene una transición positiva, su efecto se bloqueará. Sólo si RST = 1 podrá el suceso de reloj **posedge** transferir sincrónicamente D a Q.

Por lo regular es necesario que los flip-flops incluyan una señal de entrada de restablecimiento (o preestablecimiento, *preset*); de lo contrario, no se podrá determinar el estado inicial del circuito secuencial. Los circuitos secuenciales no se pueden probar con simulación HDL si no es posible asignar un estado inicial con una señal de entrada.

El ejemplo HDL 5-3 describe la construcción de un flip-flop T o JK a partir de un flip-flop D y compuertas. El circuito se describe utilizando las ecuaciones características de los flip-flops:

$$Q(t+1) = Q \oplus T \qquad \text{para un flip-flop } T$$
$$Q(t+1) = JQ' + K'Q \qquad \text{para un flip-flop } JK$$

Ejemplo HDL 5-3

```
//Flip-flop T hecho con flip-flop D y compuertas
module TFF (Q,T,CLK,RST);
   output Q;
   input T,CLK,RST;
   wire DT;
   assign DT = Q ^ T ;
//Crear ejemplar del flip-flop D
   DFF TF1 (Q,DT,CLK,RST);
endmodule

//Flip-flop JK hecho con flip-flop D y compuertas
module JKFF (Q,J,K,CLK,RST);
   output Q;
   input J,K,CLK,RST;
   wire JK;
   assign JK = (J & ~Q) | (~K & Q);
//Crear ejemplar de flip-flop D
   DFF JK1 (Q,JK,CLK,RST);
endmodule

//Flip-flop D
module DFF (Q,D,CLK,RST);
   output Q;
   input D,CLK,RST;
   reg Q;
   always @ (posedge CLK or negedge RST)
     if (~RST) Q = 1'b0;
     else Q = D;
endmodule
```

Ejemplo HDL 5-4

```
//Descripción funcional de flip-flop JK
module  JK_FF (J,K,CLK,Q,Qnot);
   output Q,Qnot;
   input  J,K,CLK;
   reg  Q;
   assign Qnot = ~ Q ;
   always @ (posedge CLK)
           case ({J,K})
             2'b00: Q = Q;
             2'b01: Q = 1'b0;
             2'b10: Q = 1'b1;
             2'b11: Q = ~ Q;
           endcase
endmodule
```

El primer módulo TFF describe un flip-flop *T* creando un ejemplar de DFF (la creación de ejemplares se explica en la sección 4-11). A la declaración **wire** DT se asigna el OR exclusivo de *Q* y *T*, lo cual es necesario para convertir un flip-flop *D* en un flip-flop *T*. La creación de un ejemplar en la que el valor de DT sustituya a *D* en el módulo DFF produce el flip-flop *T* requerido. El flip-flop *JK* se especifica de forma similar utilizando su ecuación característica para definir lo que sustituirá a *D* en el ejemplar de DFF.

El ejemplo HDL 5-4 muestra otra forma de describir un flip-flop *JK*. Aquí optamos por describirlo empleando la tabla característica en lugar de la ecuación característica. La condición de ramificación multivía **case** examina el número de dos bits que se obtiene concatenando los bits de *J* y *K*. El valor **case** ({J,K}) se evalúa y compara con los valores de la lista de enunciados que sigue. Se ejecuta el primer valor que coincide con la condición verdadera. Puesto que la concatenación de *J* y *K* produce un número de dos bits, puede ser igual a 00, 01, 10 o 11. El primer bit da el valor de *J*, y el segundo, el de *K*. Las cuatro posibles condiciones especifican el valor del siguiente estado de *Q* después de la aplicación de un reloj de borde positivo.

Diagrama de estados

El funcionamiento de los circuitos secuenciales se describe en HDL en el mismo formato que los diagramas de estados. En el ejemplo HDL 5-5 se presenta un diagrama de estados de modelo Mealy. La entrada, salida, reloj y restablecimiento se declaran de la forma acostumbrada. El estado de los flip-flops se declara con los identificadores `Prstate` (estado actual) y `Nxtstate` (siguiente estado). Estas variables contienen el valor de estado del circuito secuencial. La asignación binaria de estado se efectúa con un enunciado de parámetro. (Verilog permite definir constantes en un módulo con la palabra clave **parameter**). Se asignan los números binarios 00 a 11 a los cuatro estados $S0$ a $S3$. La notación $S2 = 2$ ' b10 es preferible a la alternativa, $S2 = 2$. La primera usa dos bits para almacenar la constante. La segunda notación produce un número binario de 32 (o 64) bits.

La descripción HDL utiliza tres bloques **always** que se ejecutan de forma concurrente e interactúan a través de variables en común. El primer enunciado **always** restablece el circuito al estado inicial $S0 = 00$ y especifica la operación sincrónica con reloj. El enunciado

Ejemplo HDL 5-5

```
//Diagrama de estados Mealy (figura 5-16)
module Mealy_mdl (x,y,CLK,RST);
  input x,CLK,RST;
  output y;
  reg y;
  reg [1:0] Prstate, Nxtstate;
  parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;
   always @ (posedge CLK or negedge RST)
      if (~RST) Prstate = S0;  //Iniciar en estado S0
      else Prstate = Nxtstate; //Operaciones de reloj
   always @ (Prstate or x)     //Determinar siguiente estado
         case (Prstate)
            S0: if (x) Nxtstate = S1;
                  else Nxtstate = S0;
            S1: if (x) Nxtstate = S3;
                  else Nxtstate = S0;
            S2: if (~x)Nxtstate = S0;
                  else Nxtstate = S2;
            S3: if (x) Nxtstate = S2;
                  else Nxtstate = S0;
         endcase
   always @ (Prstate or x)     //Evaluar salida
         case (Prstate)
            S0: y = 0;
            S1: if (x) y = 1'b0; else y = 1'b1;
            S2: if (x) y = 1'b0; else y = 1'b1;
            S3: if (x) y = 1'b0; else y = 1'b1;
         endcase
  endmodule
```

`Prstate = Nxtstate` se ejecuta únicamente en respuesta a una transición de borde positivo del reloj. Esto implica que cualquier cambio que sufra el valor de `Nxtstate` en el segundo bloque **always** se transferirá a `Prstate` como resultado de un suceso **posedge.** El segundo bloque **always** determina la transición al siguiente estado en función del estado actual y de la entrada. La condición de ramificación multivía sigue la sucesión especificada en el diagrama de estados de la figura 5-16. El tercer bloque **always** evalúa la salida en función del estado actual y de la entrada. Aunque se presenta aparte por claridad, podría combinarse con el segundo bloque. Observe que el valor de la salida *y* podría cambiar si cambia el valor de la entrada *x* mientras el circuito está en cualquier estado dado.

En el ejemplo HDL 5-6 se describe un ejemplo de diagrama de estados de modelo Moore. Este ejemplo demuestra que es posible especificar las transiciones de estado con un solo bloque **always.** El estado actual del circuito se identifica con la variable `state`. Las transiciones de estado se presentan con el CLK **posedge** de acuerdo con las condiciones dadas en los enunciados **case.** La salida del circuito es independiente de la entrada y se toma directamente de las salidas de los flip-flops. La salida de dos bits *AB* se especifica con un enunciado **assign** y es igual al valor del estado actual.

Ejemplo HDL 5-6

```
//Diagrama de estados de Moore (figura 5-19)
module Moore_mdl (x,AB,CLK,RST);
   input x,CLK,RST;
   output [1:0]AB;
   reg [1:0] state;
   parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;
      always @ (posedge CLK or negedge RST)
         if (~RST) state = S0;  //Iniciar en estado S0
         else
         case (state)
           S0: if (~x) state = S1; else state = S0;
           S1: if (x)  state = S2; else state = S3;
           S2: if (~x) state = S3; else state = S2;
           S3: if (~x) state = S0; else state = S3;
         endcase
   assign AB = state;        //Salida de flip-flops
endmodule
```

Descripción estructural

Los circuitos combinacionales se describen en HDL empleando enunciados de flujo de datos o en el nivel de compuertas. En el caso de los circuitos secuenciales, el funcionamiento de los flip-flops se describe con enunciados de comportamiento. Puesto que un circuito secuencial consta de flip-flops y compuertas, su estructura se puede describir con una combinación de enunciados de flujo de datos y de comportamiento. Los flip-flops se describen con un enunciado **always.** La parte combinacional se describe con enunciados **assign** y ecuaciones booleanas. Los módulos individuales se pueden combinar creando ejemplares.

La descripción estructural de un circuito secuencial se ilustra en el ejemplo HDL 5-7. El ejemplo tiene dos módulos. El primero describe el circuito de la figura 5-20a). El segundo describe un flip-flop *T*. Otro módulo genera un estímulo para probar el funcionamiento del circuito. El circuito secuencial es un contador binario de dos bits controlado por la entrada *x*. La salida *y* es 1 cuando la cuenta llega al 11 binario. Se incluyen los flip-flops *A* y *B* como salidas para verificar su funcionamiento. Las ecuaciones de entrada de los flip-flops y la ecuación de salida se evalúan con enunciados **assign** que tienen las expresiones booleanas correspondientes. Luego se crean ejemplares del flip-flop *T* empleando TA y TB definidos por las ecuaciones de entrada.

El segundo módulo describe el flip-flop *T*. La entrada RST restablece el flip-flop a 0 con una señal negativa. El funcionamiento del flip-flop se especifica con su ecuación característica $Q(t + 1) = Q \oplus T$.

El módulo de estímulo alimenta entradas al circuito para verificar la respuesta de salida. El primer bloque **initial** produce ocho ciclos de reloj con un periodo de 10 ns. El segundo bloque **initial** especifica un cambio alterno de la entrada *x* que se presenta en la transición de borde negativo del reloj. El resultado de la simulación se presenta en la figura 5-21. Las salidas *A* y *B* pasan por la sucesión binaria 00, 01, 10, 11 y de vuelta a 00. El cambio en el conteo se da durante un borde positivo del reloj siempre que $x = 1$. Si $x = 0$, la cuenta no cambia. La salida *y* es 1 cuando tanto *A* como *B* son 1. Esto verifica el funcionamiento del circuito.

Ejemplo HDL 5-7

```
//Descripción estructural de circuito secuencial
//Véase la figura 5-20a)
module Tcircuit (x,y,A,B,CLK,RST);
   input x,CLK,RST;
   output y,A,B;
   wire TA,TB;
//Ecuaciones de entrada de flip-flop
   assign TB = x,
          TA = x & B;
//Ecuación de salida
   assign y = A & B;
//Se crean ejemplares de flip-flops T
   T_FF BF (B,TB,CLK,RST);
   T_FF AF (A,TA,CLK,RST);
endmodule

//Flip-flop T
module T_FF (Q,T,CLK,RST);
   output Q;
   input T,CLK,RST;
   reg Q;
     always @ (posedge CLK or negedge RST)
       if (~RST) Q = 1'b0;
       else Q = Q ^ T;
endmodule

//Estímulo para probar el circuito secuencial
module testTcircuit;
  reg x,CLK,RST;  //entradas del circuito
  wire y,A,B;    //salida del circuito
  Tcircuit TC (x,y,A,B,CLK,RST);  // se crea un ejemplar
                                  // del circuito
  initial
     begin
         RST = 0;
         CLK = 0;
      #5 RST = 1;
         repeat (16)
      #5 CLK = ~CLK;
     end
  initial
     begin
          x = 0;
      #15 x = 1;
          repeat (8)
      #10 x = ~ x;
     end
endmodule
```

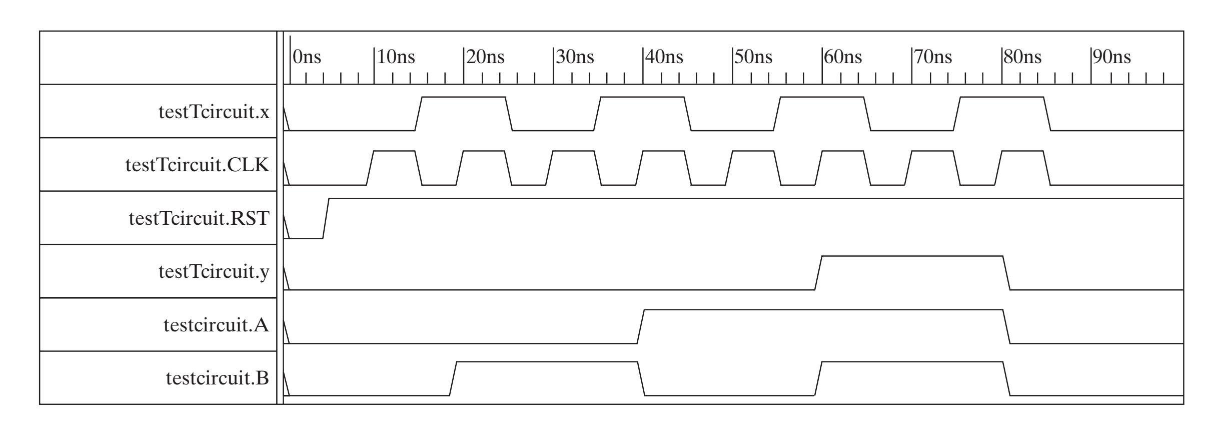


FIGURA 5-21
Salida de la simulación del ejemplo HDL 5-7

5-6 REDUCCIÓN Y ASIGNACIÓN DE ESTADOS

El análisis de circuitos secuenciales parte de un diagrama de circuitos y culmina en una tabla o diagrama de estados. El diseño de un circuito secuencial parte de un conjunto de especificaciones y culmina en un diagrama lógico. Presentaremos los procedimientos de diseño en la sección 5-7. En esta sección veremos ciertas propiedades de los circuitos secuenciales que podrían servir para reducir el número de compuertas y flip-flops durante el diseño.

Reducción de estados

La reducción en el número de flip-flops de un circuito secuencial se conoce como problema de *reducción de estados*. Los algoritmos de reducción de estados dan pie a procedimientos para reducir el número de estados de una tabla de estados, pero sin alterar los requisitos externos de entrada-salida. Puesto que m flip-flops producen 2^m estados, una reducción en el número de estados podría (o no) reducir el número de flip-flops. Un efecto impredecible al reducir el número de flip-flops es que a veces el circuito equivalente (con menos flip-flops) podría requerir más compuertas combinacionales.

Ilustraremos el procedimiento de reducción de estados con un ejemplo. Partiremos de un circuito secuencial cuya especificación se da en el diagrama de estados de la figura 5-22. En este ejemplo, sólo son importantes las sucesiones de entrada-salida; los estados internos sólo sirven para producir las sucesiones requeridas. Por ello, los estados marcados dentro de los círculos se denotan con letras en vez de sus valores binarios. Esto contrasta con un contador binario, en el que la sucesión de valores binarios de los estados mismos se toma como las salidas.

Hay un número infinito de sucesiones de entrada que podrían aplicarse al circuito; cada una produce una sucesión de salida única. Por ejemplo, consideremos la sucesión de entrada 01010110100 partiendo del estado inicial a. Cada entrada de 0 o 1 produce una salida de 0 o 1 y hace que el circuito pase al siguiente estado. Del diagrama de estados, obtenemos las su-

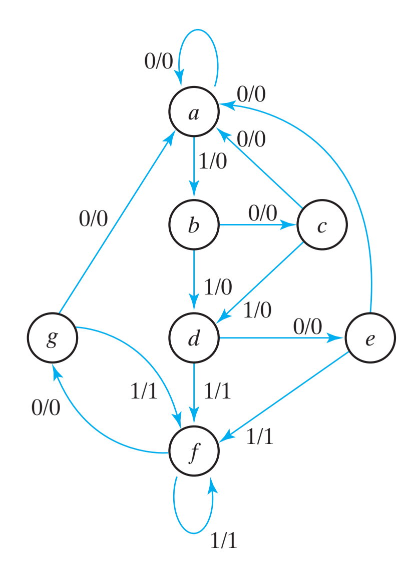


FIGURA 5-22
Diagrama de estados

cesiones de salida y de estados para la sucesión dada de entrada como sigue: con el circuito en el estado inicial *a*, una entrada de 0 produce una salida de 0 y el circuito permanece en el estado *a*. Con estado actual *a* y entrada de 1, la salida es 0 y el siguiente estado es *b*. Con estado actual *b* y entrada de 0, la salida es 0 y el siguiente estado es *c*. Continuando este proceso, se obtiene la sucesión total siguiente:

Estado	*a*	*a*	*b*	*c*	*d*	*e*	*f*	*f*	*g*	*f*	*g*	*a*
Entrada	0	1	0	1	0	1	1	0	1	0	0	
Salida	0	0	0	0	0	1	1	0	1	0	0	

En cada columna, tenemos el estado actual, el valor de entrada y el valor de salida. El siguiente estado aparece hasta arriba en la siguiente columna. Es importante darse cuenta de que, en este circuito, los estados mismos tienen importancia secundaria porque únicamente nos interesan las sucesiones de salida causadas por sucesiones de entrada.

Suponga ahora que hemos hallado un circuito secuencial cuyo diagrama de estados tiene menos de siete estados y queremos compararlo con el circuito cuyo diagrama de estados está dado por la figura 5-22. Si se aplican sucesiones de entrada idénticas a los dos circuitos y se producen salidas idénticas para todas las sucesiones de entrada, decimos que los dos circuitos son equivalentes (en lo que se refiere a entrada-salida), y podemos sustituir uno por el otro. El problema de la reducción de estados consiste en hallar formas de reducir el número de estados de un circuito secuencial sin alterar las relaciones de entrada-salida.

Ahora procedemos a reducir el número de estados de este ejemplo. Primero, necesitamos la tabla de estados; es más fácil aplicar los procedimientos de reducción de estados a una tabla que a un diagrama. La tabla de estados del circuito aparece en la tabla 5-6 y se obtiene directamente del diagrama de estados.

Tabla 5-6
Tabla de estados

Estado actual	Siguiente estado		Salida	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	*a*	*b*	0	0
b	*c*	*d*	0	0
c	*a*	*d*	0	0
d	*e*	*f*	0	1
e	*a*	*f*	0	1
f	*g*	*f*	0	1
g	*a*	*f*	0	1

Presentaremos aquí, sin demostrarlo, un algoritmo para reducir los estados de una tabla de estados plenamente especificada: "Decimos que dos estados son equivalentes si, para cada miembro del conjunto de entradas, dan exactamente la misma salida y pasan el circuito al mismo estado o a un estado equivalente". Si dos estados son equivalentes, uno de ellos puede eliminarse sin alterar las relaciones de entrada-salida.

Apliquemos ahora este algoritmo a la tabla 5-6. Examinamos la tabla en busca de estados actuales que pasen al mismo siguiente estado y tengan la misma salida con ambas combinaciones de entrada. Los estados *g* y *e* cumplen con esos requisitos: ambos pasan a los estados *a* y *f* y tienen salidas de 0 y 1 con $x = 0$ y $x = 1$, respectivamente. Por tanto, los estados *g* y *e* son equivalentes y podemos eliminar uno de ellos. En la tabla 5-7 se indica el procedimiento para eliminar un estado y sustituirlo por su equivalente. Se elimina la fila del estado actual *g* y el estado *g* se sustituye por *e* en todos los lugares en que aparece en las columnas de siguiente estado.

El estado actual *f* ahora tiene como siguientes estados a *e* y *f*, y como salidas, 0 y 1 cuando $x = 0$ y $x = 1$, respectivamente. En la fila del estado actual *d* aparecen los mismos siguientes estados y salidas. Por tanto, los estados *f* y *d* son equivalentes y podemos eliminar el estado *f*, sustituyéndolo por *d*. La tabla reducida final se reproduce en la tabla 5-8. El diagrama de estados de la tabla reducida consta únicamente de cinco estados y se aprecia en la

Tabla 5-7
Reducción de la tabla de estados

Estado actual	Siguiente estado		Salida	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	*a*	*b*	0	0
b	*c*	*d*	0	0
c	*a*	*d*	0	0
d	*e*	*f*	0	1
e	*a*	*f*	0	1
f	*e*	*f*	0	1

Tabla 5-8
Tabla de estados reducida

	Siguiente estado		Salida	
Estado actual	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1

figura 5-23. Este diagrama de estados satisface las especificaciones de entrada-salida originales y produce la sucesión de salida requerida con cualquier sucesión de entrada dada. La lista que sigue se dedujo del diagrama de estados de la figura 5-23 y corresponde a la sucesión de entrada que se utilizó antes (advierta que se obtiene la misma sucesión de salida, aunque la sucesión de estados es distinta):

Estado	a	a	b	c	d	e	d	d	e	d	e	a
Entrada	0	1	0	1	0	1	1	0	1	0	0	
Salida	0	0	0	0	0	1	1	0	1	0	0	

De hecho, esta sucesión es exactamente la misma que se obtuvo con la figura 5-21 si se sustituye g por e y f por d.

Podemos verificar sistemáticamente la posible equivalencia de cada par de estados con la ayuda de un procedimiento que utiliza una tabla de implicación. Dicha tabla tiene un cuadrado para cada par de estados que se sospecha podrían ser equivalentes. Si usamos la tabla acertadamente, podremos encontrar todos los pares de estados equivalentes de una tabla de estados. Ilustraremos el uso de la tabla de implicación para reducir el número de estados de una tabla de estados en la sección 9.5.

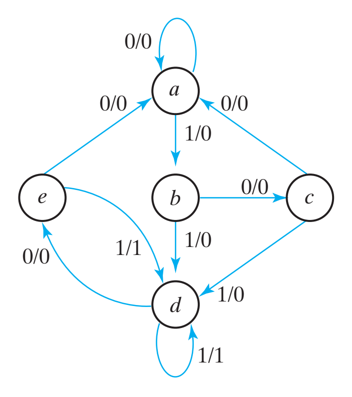


FIGURA 5-23
Diagrama de estados reducido

El circuito secuencial de este ejemplo se redujo de siete a cinco estados. En general, la reducción del número de estados de una tabla de estados da pie a un circuito con menos componentes. No obstante, el hecho de que una tabla de estados se haya reducido a menos estados no garantiza un ahorro en el número de flip-flops o de compuertas.

Asignación de estados

Para diseñar un circuito secuencial con componentes físicos, es necesario asignar valores binarios codificados a los estados. En el caso de un circuito con m estados, los códigos deben contener n bits, donde $2^n = \geq m$. Por ejemplo, con tres bits es posible asignar códigos a ocho estados denotados por los números binarios de 000 a 111. Si usamos la tabla de estados de la tabla 5-6, deberemos asignar valores binarios a siete estados; el estado restante no se usa. Si utilizamos la tabla de estados de la tabla 5-8, sólo cinco estados requerirán asignación binaria, y nos quedarán tres estados no utilizados. Los estados no utilizados se tratan como condiciones de indiferencia durante el diseño. Dado que las condiciones de indiferencia por lo regular ayudan a obtener un circuito más sencillo, es más probable que el circuito con cinco estados requiera menos compuertas combinacionales que el circuito con siete estados.

La forma más sencilla de codificar cinco estados es usar los primeros cinco enteros en el orden del conteo binario, como se muestra en la primera asignación de la tabla 5-9. Otra asignación similar es el código Gray que se muestra como asignación 2. En este caso, sólo un bit del grupo de código cambia al pasar de un número al siguiente. Este código facilita la colocación de las funciones booleanas en el mapa para simplificarlas. Otra posible asignación que se usa a menudo en el diseño de control es la asignación de un solo uno (*one-hot*). Esta configuración utiliza tantos bits como estados hay en el circuito. En cualquier momento, sólo un bit es 1; todos los demás son 0. Este tipo de asignación utiliza un flip-flop por estado.

La tabla 5-10 es la tabla de estados reducida, después de sustituir los símbolos de letra de los estados por la asignación binaria 1. Una asignación distinta producirá una tabla de estados con valores binarios distintos para los estados. Usamos la forma binaria de la tabla de estados para deducir la parte combinacional del circuito secuencial. La complejidad del circuito combinacional dependerá de la asignación binaria de estados que se escoja.

A veces se usa el término *tabla de transiciones* para referirse a una tabla de estados con asignación binaria. Esto la distingue de las tablas de estados que usan nombres simbólicos para los estados. En este libro usaremos el mismo término para referirnos a ambos tipos de tablas de estados.

Tabla 5-9
Tres posibles asignaciones binarias de estados

Estado	Asignación 1 Binaria	Asignación 2 Código Gray	Asignación 3 Un solo uno
a	000	000	00001
b	001	001	00010
c	010	011	00100
d	011	010	01000
e	100	110	10000

Tabla 5-10
Tabla de estados reducida, con la asignación binaria 1

Estado actual	Siguiente estado $x = 0$	Siguiente estado $x = 1$	Salida $x = 0$	Salida $x = 1$
000	000	001	0	0
001	010	011	0	0
010	000	011	0	0
011	100	011	0	1
100	000	011	0	1

5-7 PROCEDIMIENTO DE DISEÑO

El diseño de un circuito secuencial con reloj parte de un conjunto de especificaciones y culmina en un diagrama lógico o una lista de funciones booleanas de la cual puede obtenerse el diagrama lógico. En contraste con los circuitos combinacionales, que se especifican cabalmente con una tabla de verdad, los circuitos secuenciales requieren una tabla de estados para su especificación. El primer paso en el diseño de circuitos secuenciales es la obtención de una tabla de estados o una representación equivalente, como un diagrama de estados.

Un circuito secuencial sincrónico consta de flip-flops y compuertas combinacionales. El diseño del circuito consiste en escoger los flip-flops y luego encontrar una estructura de compuertas combinacionales que, junto con los flip-flops, produzca un circuito que satisfaga las especificaciones planteadas. El número de flip-flops se deduce del número de estados que se requieren en el circuito. El circuito combinacional se deduce de la tabla de estados evaluando las ecuaciones de entrada y de salida de los flip-flops. De hecho, una vez determinados el tipo y el número de los flip-flops, el proceso de diseño implica una transformación de un problema de circuito secuencial a un problema de circuito combinacional. De este modo, pueden aplicarse las técnicas del diseño de circuitos combinacionales.

El procedimiento para diseñar circuitos secuenciales sincrónicos se resume en una lista de pasos recomendados:

1. Deduzca, de la descripción textual y las especificaciones del funcionamiento deseado, un diagrama de estados para el circuito.
2. Reduzca el número de estados si es necesario.
3. Asigne valores binarios a los estados.
4. Obtenga la tabla de estados codificada en binario.
5. Escoja el tipo de flip-flops que se usarán.
6. Deduzca las ecuaciones simplificadas de entrada y de salida de los flip-flops.
7. Dibuje el diagrama lógico.

La especificación textual del comportamiento del circuito por lo regular supone que el lector conoce la terminología de lógica digital. Es necesario que el diseñador utilice intuición y experiencia para interpretar correctamente las especificaciones del circuito, porque las descripciones textuales podrían ser incompletas e inexactas. Una vez establecida tal especificación, y habiéndose obtenido el diagrama de estados, será posible aplicar procedimientos conocidos de síntesis para completar el diseño. Aunque existen procedimientos formales para la reducción y asignación de estados, los diseñadores experimentados casi nunca los usan. Los pasos

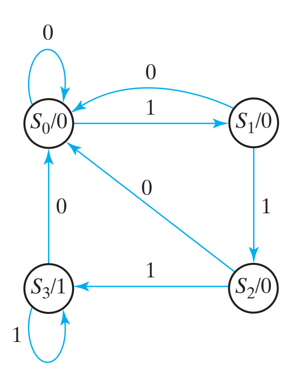


FIGURA 5-24
Diagrama de estados para el detector de sucesiones

4 a 7 del diseño se implementan con algoritmos exactos y por tanto pueden automatizarse. La parte del diseño que sigue un procedimiento bien definido se denomina *síntesis*.

El primer paso es la parte más difícil del diseño. Aquí mostraremos un ejemplo sencillo para ilustrar la forma de obtener un diagrama de estados a partir de la especificación textual.

Queremos diseñar un circuito que detecte tres o más unos consecutivos en una cadena de bits que llegan por una línea de entrada. El diagrama de estados del circuito se presenta en la figura 5-24. Se obtiene partiendo del estado S_0. Si la entrda es 0, el circuito permanece en el mismo estado, pero si es 1, pasa al estado S_1 para indicar que se detectó un 1. Si la siguiente entrada es 1, el cambio es al estado S_2, para indicar que han llegado dos unos consecutivos, pero si la entrada es 0 volvemos al estado S_0. El tercer uno consecutivo envía al circuito al estado S_3. Si se detectan más unos, el circuito permanecerá en S_3. Cualquier entrada 0 devolverá el circuito a S_0. Así, el circuito permanecerá en S_3 en tanto se hayan recibido tres o más unos consecutivos. Se trata de un circuito secuencial de modelo Moore porque la salida es 1 cuando el circuito está en el estado S_3, y 0 en los demás casos.

Síntesis con flip-flops

Una vez deducido el diagrama de estados, el resto del diseño sigue un procedimiento de síntesis directo. De hecho, es posible diseñar el circuito con una descripción HDL del diagrama de estados y las herramientas de síntesis HDL apropiadas para obtener una lista de red sintetizada. (La descripción HDL del diagrama de estados será similar al ejemplo HDL 5-6 de la sección 5-5.) Para diseñar el circuito a mano, necesitamos asignar códigos binarios a los estados y preparar la tabla de estados. Esto se hizo en la tabla 5-11, que se dedujo del diagrama de estados de la figura 5-24 con una asignación binaria directa. Escogimos dos flip-flops *D* para representar los cuatro estados y rotulamos sus salidas *A* y *B*. Hay una entrada *x* y una salida *y*. La ecuación característica del flip-flop *D* es $Q(t+1) = D_Q$, lo que significa que los valores de siguiente estado de la tabla de estados especifican la condición de entrada *D* para el flip-flop. Las ecuaciones de entrada del flip-flop se obtienen directamente de las columnas de siguiente estado de *A* y *B*, y se expresan en forma de suma de minitérminos así:

$$A(t+1) = D_A(A, B, x) = \sum(3, 5, 7)$$
$$B(t+1) = D_B(A, B, x) = \sum(1, 5, 7)$$
$$y(A, B, x) = \sum(6, 7)$$

Tabla 5-11
Tabla de estados para el detector de sucesiones

Estado actual		Entrada	Siguiente Estado		Salida
A	*B*	*x*	*A*	*B*	*y*
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	1

donde A y B son los valores de estado actual de los flip-flops A y B, x es la entrada, y D_A y D_B son las ecuaciones de entrada. Los minitérminos para la salida y se obtienen de la columna de salida de la tabla de estados.

Las ecuaciones booleanas se simplifican con ayuda de los mapas de la figura 5-25. Las ecuaciones simplificadas son

$$D_A = Ax + Bx$$
$$D_B = Ax + B'x$$
$$y = AB$$

El diagrama lógico del circuito secuencial se presenta en la figura 5-26.

Tablas de excitación

El diseño de un circuito secuencial con flip-flops de otro tipo que no sea D se complica por el hecho de que las ecuaciones de entrada del circuito se deben deducir de manera indirecta de la tabla de estados. Cuando se usan flip-flops D, las ecuaciones de entrada se obtienen direc-

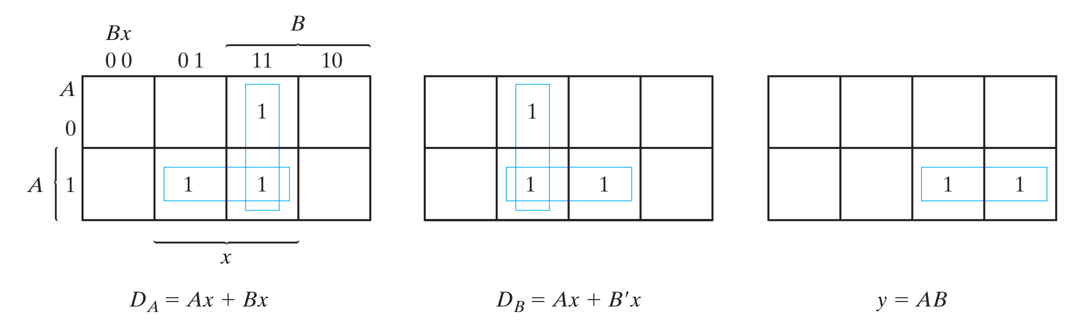


FIGURA 5-25
Mapas para el detector de sucesiones

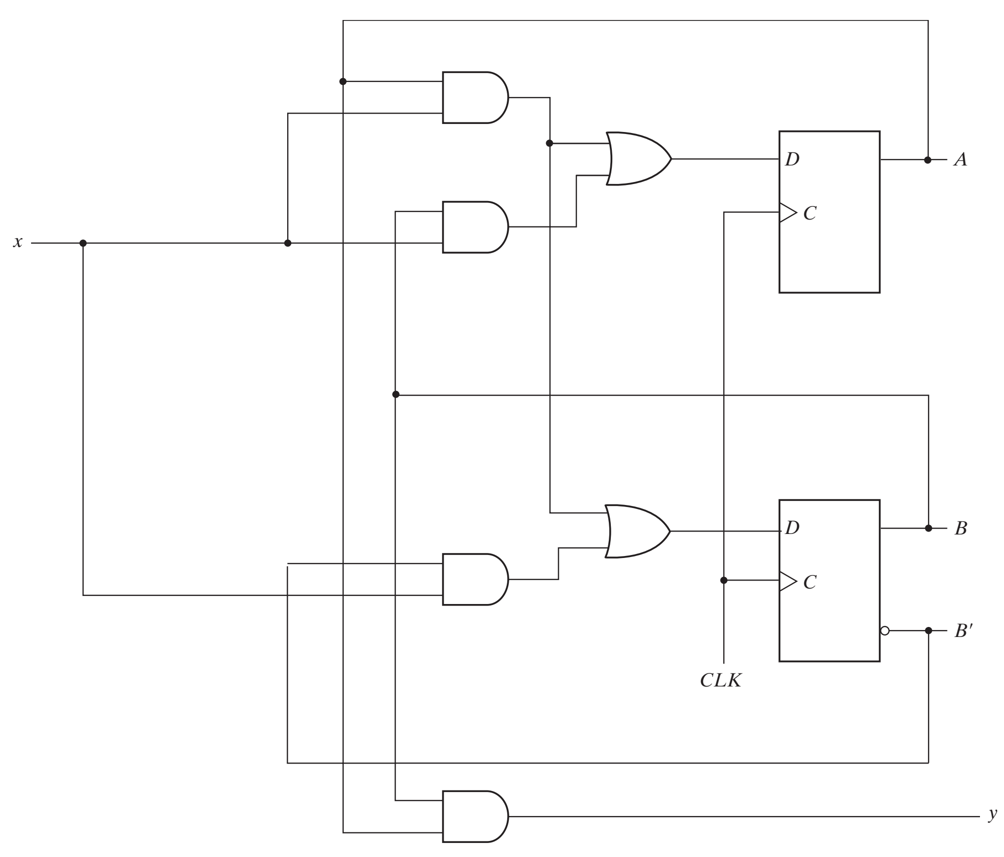


FIGURA 5-26
Diagrama lógico del detector de sucesiones

tamente del siguiente estado. No sucede así con los flip-flops *JK* y *T*. Para determinar las ecuaciones de entrada de estos flip-flops, es necesario deducir una relación funcional entre la tabla de estados y las ecuaciones de entrada.

Las tabla características de flip-flops que presentamos en la tabla 5-1 dan el valor del siguiente estado cuando se conocen las entradas y el estado actual. Estas tablas son útiles para analizar circuitos secuenciales y para definir el funcionamiento de los flip-flops. Durante el proceso de diseño, normalmente se conoce la transición de estado actual a siguiente estado y se desea conocer las condiciones de entrada del flip-flop que dan pie a la transición requerida. Por ello, se necesita una tabla que dé las entradas requeridas para un cambio de estado dado. Ese tipo de tablas se llaman *tablas de excitación*.

La tabla 5-12 presenta las tablas de excitación de los dos flip-flops. Cada tabla tiene una columna para el estado actual, $Q(t)$ y el siguiente estado, $Q(t + 1)$, y una columna para cada entrada, a fin de mostrar cómo se logra la transición requerida. Hay cuatro posibles transiciones de estado actual a siguiente estado. Las condiciones de entrada necesarias para cada una se deducen de la información proporcionada por la tabla característica. El símbolo X en las tablas representa una condición de indiferencia, es decir, que no importa si la entrada es 1 o 0.

La tabla de excitación del flip-flop *JK* corresponde a la parte a). Cuando tanto el estado actual como el siguiente son 0, la entrada *J* debe permanecer en 0 y la entrada *K* puede ser 0 o 1.

Tabla 5-12
Tablas de excitación de flip-flops

Q(t)	Q(t + 1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

a) JK

Q(t)	Q(t + 1)	T
0	0	0
0	1	1
1	0	1
1	1	0

b) T

Asimismo, cuando tanto el estado actual como el siguiente son 1, la entrada K debe permanecer en 0, mientras que la J puede ser 0 o 1. Si es preciso que el flip-flop tenga una transición del estado 0 al 1, J deberá ser 1, porque la entrada J establece el flip-flop; en cambio, la entrada K puede ser 0 o 1. Si $K = 0$, la condición $J = 1$ establecerá el flip-flop como se requiere; si $K = 1$ y $J = 1$, el flip-flop se complementará y pasará del estado 0 al 1, como se requiere. Por tanto, marcamos la entrada K con una condición de indiferencia para la transición de 0 a 1. En el caso de la transición del estado 1 al 0, necesitamos $K = 1$, porque la entrada K despeja el flip-flop. En cambio, la entrada J puede ser 0 o 1, porque $J = 0$ no tiene ningún efecto, mientras que $J = 1$ con $K = 1$ complementa el flip-flop y produce la transición del estado 1 al estado 0.

La parte b) corresponde a la tabla de excitación para el flip-flop T. Por la tabla de características, sabemos que, cuando la entrada $T = 1$, el estado del flip-flop se complementa; cuando $T = 0$, el estado del flip-flop no cambia. Por tanto, si el estado del flip-flop debe permanecer sin cambio, el requisito es que $T = 0$. Si es preciso complementar el estado del flip-flop, T debe ser 1.

Síntesis con flip-flops *JK*

El procedimiento de síntesis de circuitos secuenciales con flip-flops JK es el mismo que con flip-flops D, excepto que las ecuaciones de entrada se deben evaluar a partir de la transición de estado actual a siguiente estado deducida de la tabla de excitación. Para ilustrar el procedimiento, sintetizaremos el circuito secuencial especificado por la tabla 5-13. Además de tener columnas para el estado actual, la entrada y el siguiente estado, como en una tabla convencional de estados, la tabla también muestra las condiciones de entrada del flip-flop de las que se deducen las ecuaciones de entrada. Estas entradas del flip-flop se deducen de la tabla de estados y de la tabla de excitación del flip-flop JK. Por ejemplo, en la primera fila de la tabla 5-13 tenemos una transición para el flip-flop A, de 0 en el estado actual a 0 en el siguiente estado. En la tabla 5-12 para el flip-flop JK, vemos que una transición de estado actual 0 a siguiente estado 0 requiere que la entrada J sea 0; la entrada K no importa. Por tanto, se anota un 0 y una X en la primera fila, bajo J_A y K_A. Puesto que la primera fila también muestra una transición para el flip-flop B, de 0 en el estado actual a 0 en el siguiente estado, se anota un 0 y una X en esa fila, bajo J_B y K_B. La segunda fila de la tabla indica una transición del flip-flop B, de 0 en el estado actual a 1 en el siguiente estado. La tabla de excitación nos dice que una transición de 0 a 1 requiere $J = 1$, mientras que K no importa, así que anotamos 1 y X en la segunda fila bajo J_B y K_B. Continuamos este proceso con todas las filas de la tabla y con cada flip-flop, copiando las condiciones de entrada de la tabla de excitación en la fila apropiada del flip-flop que se está considerando.

Las entradas del flip-flop de la tabla 5-13 especifican la tabla de verdad de las ecuaciones de entrada en función de los estados actuales A y B y de la entrada x. Las ecuaciones de entrada se simplifican en los mapas de la figura 5-27. Los valores de siguiente estado no se usan duran-

Tabla 5-13
Tabla de estados y entradas de flip-flops JK

Estado Actual		Entrada	Siguiente Estado		Entradas del flip-flop			
A	B	x	A	B	J_A	K_A	J_B	K_B
0	0	0	0	0	0	X	0	X
0	0	1	0	1	0	X	1	X
0	1	0	1	0	1	X	X	1
0	1	1	0	1	0	X	X	0
1	0	0	1	0	X	0	0	X
1	0	1	1	1	X	0	1	X
1	1	0	1	1	X	0	X	0
1	1	1	0	0	X	1	X	1

te la simplificación porque las ecuaciones de entrada son función únicamente del estado actual y de la entrada. Considere la ventaja de usar flip-flops JK al diseñar circuitos secuenciales. El hecho de que haya tantas condiciones de indiferencia indica que el circuito combinacional para las ecuaciones de entrada seguramente será más sencillo, porque los minitérminos de indiferencia normalmente ayudan a obtener expresiones más simples. Si la tabla de estados tiene estados no utilizados, habrá condiciones de indiferencia adicionales en el mapa.

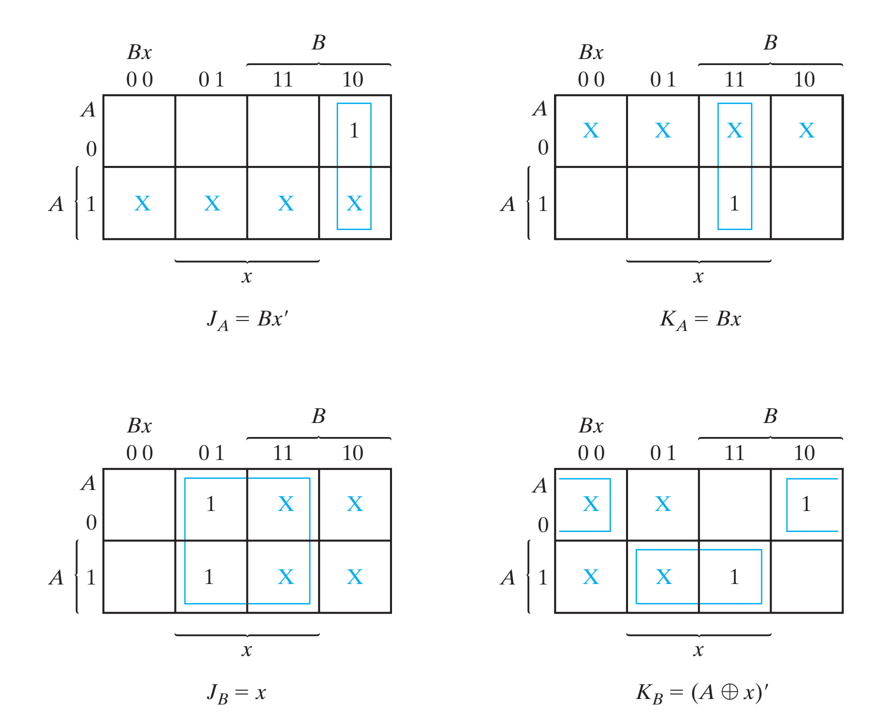


FIGURA 5-27
Mapas para las ecuaciones de entrada *J* y *K*

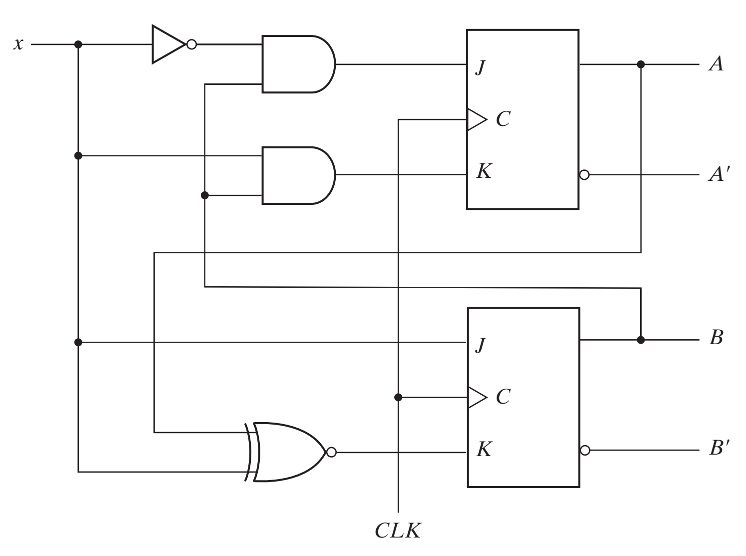


FIGURA 5-28
Diagrama lógico para el circuito secuencial con flip-flops *JK*

Las cuatro ecuaciones de entrada de los dos flip-flops *JK* se dan bajo los mapas de la figura 5-27. El diagrama lógico del circuito secuencial se presenta en la figura 5-28.

Síntesis con flip-flops *T*

Ilustraremos la síntesis con flip-flops *T* diseñando un contador binario. Un contador binario de *n* bits consiste en *n* flip-flops capaces de contar en binario de 0 hasta $2^n - 1$. El diagrama de estados de un contador de tres bits se reproduce en la figura 5-29. Como se ve por los estados binarios indicados dentro de los círculos, las salidas de los flip-flops repiten la sucesión de conteo binario, volviendo a 000 después de 111. Las flechas entre los círculos no se han marcado con valores de entrada y salida como en otros diagramas de estados. Recuerde que las transiciones de estado en los circuitos secuenciales con reloj se dan durante un borde de reloj; los flip-flops permanecen en su estado actual si no se aplica reloj. Por ello, el reloj no aparece explícitamente como variable de entrada en el diagrama de estados ni en la tabla de estados. Desde este

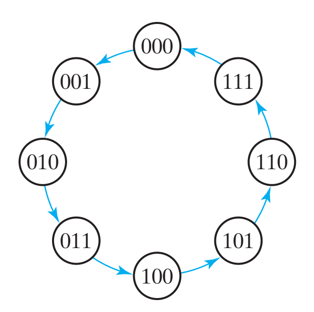


FIGURA 5-29
Diagrama de estados de un contador binario de tres bits

Tabla 5-14
Tabla de estados para el contador de tres bits

Estado actual			Siguiente estado			Entradas de los flip-flops		
A_2	A_1	A_0	A_2	A	A_0	T_{A2}	T_{A1}	T_{A0}
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	1	1
1	1	1	0	0	0	1	1	1

punto de vista, el diagrama de estados de un contador no tiene que indicar valores de entrada y salida a lo largo de las flechas. La única entrada del circuito es el reloj, y las salidas están especificadas por el estado actual de los flip-flops. El siguiente estado de un contador depende exclusivamente de su estado actual, y la transición de estado se efectúa cada vez que el reloj tiene una transición.

La tabla 5-14 es la tabla de estados para el contador binario de tres bits. Los tres flip-flops se designan con A_2, A_1 y A_0. La forma más eficiente de construir contadores binarios es con flip-flops T gracias a su propiedad de complemento. La excitación de los flip-flops para las entradas T se deduce de la tabla de excitación del flip-flop T y de una inspección de la transición del estado actual al siguiente estado. Por ejemplo, consideremos las entradas de flip-flop que van en la fila 001. El estado actual aquí es 001, y el siguiente, 010, que es el siguiente conteo sucesivo. Si comparamos estos dos conteos, veremos que A_2 pasa de 0 a 0; por tanto, marcamos T_{A2} con 0 porque el flip-flop A_2 no debe cambiar cuando hay una transición de reloj. A_1 pasa de 0 a 1, así que marcamos T_{A1} con 1 porque este flip-flop se deberá complementar en el siguiente borde de reloj. Por su parte, A_0 pasa de 1 a 0, lo que indica que se debe complementar; por tanto, marcamos T_{A0} con 1. La última fila, con estado actual 111, se compara con el primer conteo, 000, que es su siguiente estado. El cambio de ceros a unos en todos los bits requiere complementar los tres flip-flops.

Las ecuaciones de entrada de los flip-flops se simplifican con los mapas de la figura 5-30. Vemos que T_{A0} tiene unos en los ocho minitérminos porque el bit menos significativo del con-

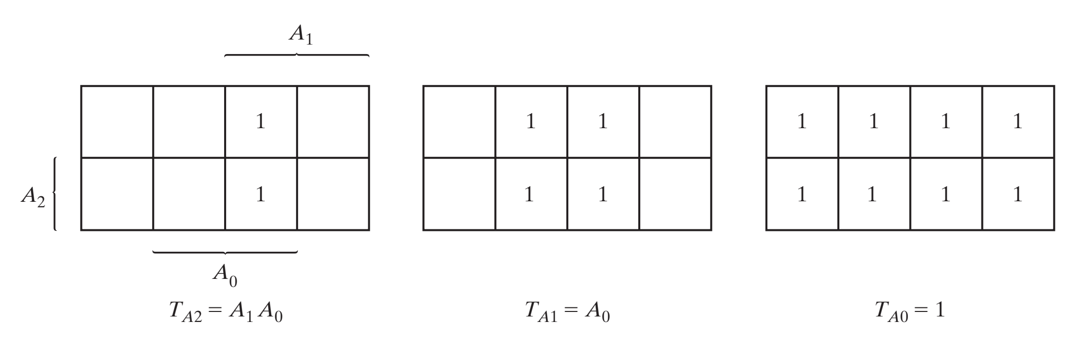


FIGURA 5-30
Mapas para el contador binario de tres bits

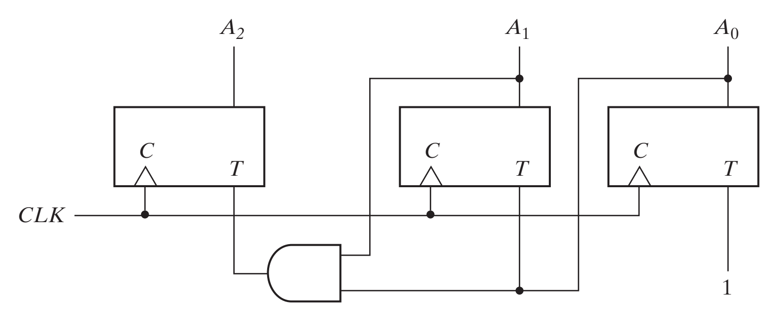


FIGURA 5-31
Diagrama lógico del contador de tres bits

tador se complementa en cada conteo. Una función booleana que incluye a todos los minitérminos define un valor constante de 1. Las ecuaciones de entrada que se dan abajo de cada mapa especifican la parte combinacional del contador. Al incluir estas funciones con los tres flip-flops, obtenemos el diagrama lógico del contador de la figura 5-31.

PROBLEMAS

5-1 El latch *D* de la figura 5-6 se construyó con cuatro compuertas NAND y un inversor. Considere estas otras tres formas de obtener un latch *D*. En cada caso, dibuje el diagrama lógico y verifique el funcionamiento del circuito.

a) Use compuertas NOR para la parte de latch *SR* y compuertas AND para las otras dos. Se podría necesitar un inversor.

b) Use compuertas NOR para las cuatro compuertas. Se podrían requerir inversores.

c) Use únicamente cuatro compuertas NAND (sin inversor). Esto se logra conectando la salida de la compuerta superior de la figura 5-6 (que va al latch *SR*) con la entrada de la compuerta inferior (en vez de la salida del inversor).

5-2 Construya un flip-flop *JK* con un flip-flop *D*, un multiplexor de 2 líneas a 1 y un inversor.

5-3 Demuestre que la ecuación característica para la salida de complemento de un flip-flop *JK* es

$$Q'(t + 1) = J'Q' + KQ$$

5-4 Un flip-flop *PN* tiene cuatro operaciones: despeje a 0, ningún cambio, complemento y establecimiento a 1, cuando las entradas *P* y *N* son 00, 01, 10 y 11, respectivamente.

a) Tabule la tabla de características.

b) Deduzca la ecuación característica.

c) Tabule la tabla de excitación.

d) Muestre cómo el flip-flop *PN* se puede convertir en un flip-flop *D*.

5-5 Explique la diferencia entre tabla de verdad, tabla de estados, tabla característica y tabla de excitación. Explique también la diferencia entre una ecuación booleana, una ecuación de estado, una ecuación característica y una ecuación de entrada de flip-flop.

5-6 Un circuito secuencial con dos flip-flops *D*, *A* y *B*; dos entradas, *x* y *y*; y una salida, *z*, se especifica con las ecuaciones de estado y salida siguientes

$$A(t + 1) = x'y + xA$$
$$B(t + 1) = x'B + xA$$
$$z = B$$

a) Dibuje el diagrama lógico del circuito.

b) Prepare la tabla de estados del circuito secuencial.

c) Dibuje el diagrama de estados correspondiente.

5-7 Un circuito secuencial tiene un flip-flop *Q*, dos entradas *x* y *y*, y una salida *S*. Consta de un circuito sumador completo conectado a un flip-flop *D*, como se indica en la figura P5-7. Deduzca la tabla de estados y el diagrama de estados del circuito secuencial.

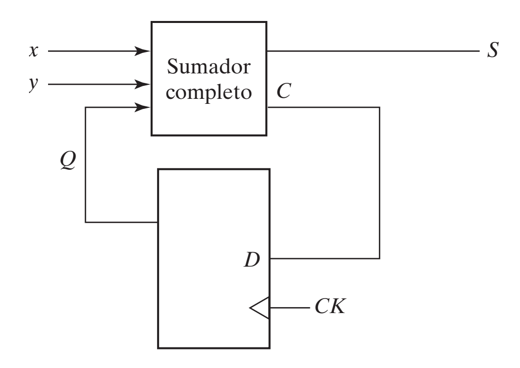


FIGURA P5-7

5-8 Deduzca la tabla de estados y el diagrama de estados del circuito secuencial que se muestra en la figura P5-8. Explique la función del circuito.

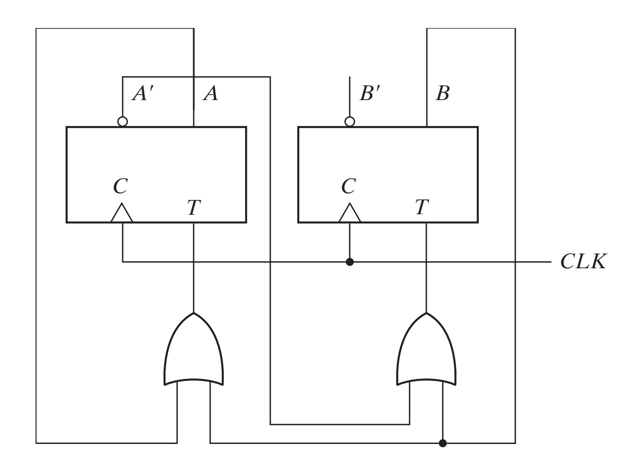


FIGURA P5-8

5-9 Un circuito secuencial tiene dos flip-flops JK, A y B, y una entrada, x. El circuito se describe con estas ecuaciones de entrada de flip-flop:

$$J_A = x \qquad K_A = B'$$
$$J_B = x \qquad K_B = A$$

a) Deduzca las ecuaciones de estado $A(t + 1)$ y $B(t + 1)$ sustituyendo las ecuaciones de entrada por las variables J y K.

b) Dibuje el diagrama de estados del circuito.

5-10 Un circuito secuencial tiene dos flip-flops JK, A y B, dos entradas, x y y, y una salida, z. Las ecuaciones de entrada de los flip-flops y la ecuación de salida del circuito son

$$J_A = Bx + B'y' \qquad K_A = B'xy'$$
$$J_B = A'x \qquad K_B = A + xy'$$
$$z = Ax'y' + Bx'y'$$

a) Dibuje el diagrama lógico del circuito. b) Prepare la tabla de estados.

c) Deduzca las ecuaciones de estado para A y B.

5-11 Partiendo del estado 00 en el diagrama de estados de la figura 5-16, determine las transiciones de estados y sucesión de salida que se generarán cuando se aplique la sucesión de entrada 010110111011110.

5-12 Reduzca el número de estados de la siguiente tabla de estados y tabule la tabla de estados reducida.

Estado actual	**Siguiente estado**		**Salida**	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	f	b	0	0
b	d	c	0	0
c	f	e	0	0
d	g	a	1	0
e	d	c	0	0
f	f	b	1	1
g	g	h	0	1
h	g	a	1	0

5-13 Partiendo del estado a y la sucesión de entrada 01110010011, determine la sucesión de salida de

a) la tabla de estados del problema anterior y

b) la tabla de estados reducida del problema anterior. Demuestre que se obtiene la misma sucesión de salida con ambas.

5-14 Sustituya la asignación binaria 2 de la tabla 5-9 en los estados de la tabla 5-8 y obtenga la tabla de estados binaria.

5-15 Prepare una tabla de estados para el flip-flop *JK* utilizando *Q* como estado actual y siguiente, y *J* y *K* como entradas. Diseñe el circuito secuencial especificado por la tabla de estados y demuestre que es equivalente a la figura 5-12a).

5-16 Diseñe un circuito secuencial con dos flip-flops *D*, *A* y *B*, y una entrada, *x*. Cuando $x = 0$, el estado del circuito no cambia. Cuando $x = 1$, el circuito pasa por las transiciones de estado de 00 a 01 a 11 a 10 y de vuelta a 00, y repite.

5-17 Diseñe un complementador a dos en serie con una entrada y una salida. El circuito acepta una cadena de bits de la entrada y genera el complemento a dos en la salida. El circuito se puede restablecer asincrónicamente para iniciar y terminar la operación.

5-18 Diseñe un circuito secuencial con dos flip-flops *JK*, *A* y *B*, y dos entradas, *E* y *x*. Si $E = 0$, el circuito permanece en el mismo estado sea cual sea el valor de *x*. Si $E = 1$ y $x = 1$, el circuito pasa por las transiciones de estado de 00 a 01 a 10 a 11 y de vuelta a 00, y repite. Cuando $E = 1$ y $x = 0$, el circuito pasa por las transiciones de estado de 00 a 11 a 10 a 01 y de vuelta a 00, y repite.

5-19 Un circuito secuencial tiene tres flip-flops, *A*, *B*, *C*; una entrada, *x*; y una salida, *y*. El diagrama de estados aparece en la figura P5-19. El circuito se diseñará tratando los estados no utilizados como condiciones de indiferencia. Analice el circuito obtenido del diseño para determinar el efecto de los estados no utilizados.

a) Use flip-flops *D* en el diseño.

b) Use flip-flops *JK* en el diseño.

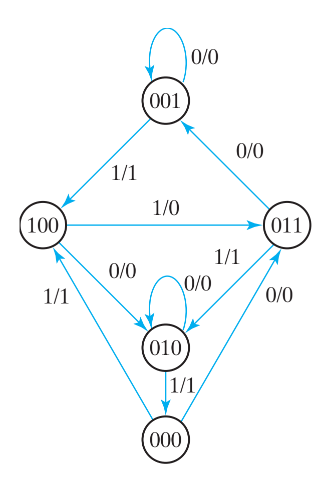


FIGURA P5-19

5-20 Diseñe el circuito secuencial especificado por el diagrama de estados de la figura 5.19 empleando flip-flops *T*.

5-21 Explique la principal diferencia entre un enunciado **initial** y un enunciado **always** en Verilog HDL.

5-22 Dibuje la forma de onda generada por el enunciado **initial**

```
initial
  begin
  w = 0;  #20 w = 1;  # 50 w = 0;  # 30 w = 1;  #10 w = 0;
  end
```

5-23 Considere estos enunciados suponiendo que RegA contiene inicialmente el valor 30.

a)
```
RegA = 125
RegB = RegA
```

b)
```
RegA <= 125
RegB <= RegA
```

¿Qué valores tienen RegA y RegB después de la ejecución?

5-24 Escriba una descripción HDL del comportamiento de un flip-flop D con preestablecimiento y restablecimiento asincrónicos. (Este tipo de flip-flop se reproduce en la figura 11-13.)

5-25 Un flip-flop especial disparado por borde positivo tiene dos entradas, $D1$ y $D2$, y una entrada de control que escoge una de las dos. Escriba una descripción HDL del comportamiento de este flip-flop.

5-26 Escriba una descripción HDL del comportamiento de un flip-flop JK utilizando un enunciado **if-else** basado en el valor del estado actual. (*Sugerencia:* Considere la ecuación característica cuando $Q = 0$ o $Q = 1$.)

5-27 Reescriba la descripción del ejemplo HDL 5-5 combinando las transiciones de estado y la salida en un bloque **always.**

5-28 Simule el circuito secuencial de la figura 5-17.

a) Escriba la descripción HDL del diagrama de estados.

b) Escriba la descripción HDL del diagrama de circuito.

c) Escriba un estímulo HDL con una sucesión de entradas: 00, 01, 11, 10. Verifique que la respuesta sea la misma con ambas descripciones.

5-29 Escriba la descripción HDL del contador binario de dos bits que se ilustra en la figura 5-20. Utilice el módulo de estímulo del ejemplo HDL 5-7 y verifique que su respuesta de salida sea la misma que las formas de onda de la figura 5-21.

5-30 Dibuje el diagrama lógico del circuito secuencial descrito por el módulo HDL siguiente:

```
module Seqcrt (A,B,C,Q,CLK);
   input A,B,C,CLK;
   output Q;
   reg Q,E;
   always @ (posedge CLK)
      begin
     E <= A & B;
     Q <= E | C;
      end
endmodule
```

¿Qué cambios, si acaso, deben hacerse al circuito si los dos últimos enunciados usan asignación bloqueadora en vez de no bloqueadora?

REFERENCIAS

1. HAYES, J. P. 1993. *Introduction to Digital Logic Design*. Reading, MA: Addison-Wesley.
2. WAKERLY, J. F. 2000. *Digital Design: Principles and Practices*, 3a. ed. Upper Saddle River, NJ: Prentice-Hall.
3. KATZ, R. H. 1994. *Contemporary Logic Design*. Upper Saddle River, NJ: Prentice-Hall.
4. MANO, M. M. y C. R. KIME. 2000. *Logic and Computer Design Fundamentals*, 2a. ed. Upper Saddle River, NJ: Prentice-Hall.
5. NELSON V. P., H. T. NAGLE, J. D. IRWIN y B. D. CARROLL. 1995. *Digital Logic Circuit Analysis and Design*. Upper Saddle River, NJ: Prentice-Hall.
6. DIETMEYER, D. L. 1988. *Logic Design of Digital Systems*, 3a. ed. Boston: Allyn Bacon.
7. GAJSKI, D. D. 1997. *Principles of Digital Design*. Upper Saddle River, NJ: Prentice-Hall.
8. ROTH, C. H. 1992. *Fundamentals of Logic Design*, 4a. ed. St. Paul: West.
9. BHASKER, J. 1997. *A Verilog HDL Primer*. Allentown, PA: Star Galaxy Press.
10. BHASKER, J. 1998. *Verilog HDL Synthesis*. Allentown, PA: Star Galaxy Press.
11. CILETTI, M. D. 1999. *Modeling, Synthesis and Rapid Prototyping with Verilog HDL*. Upper Saddle River, NJ: Prentice-Hall.
12. PALNITKAR, S. 1996. *Verilog HDL: A Guide to Digital Design and Synthesis*. SunSoft Press (un título Prentice-Hall).
13. THOMAS, D. E. y P. R. MOORBY. 1998. *The Verilog Hardware Description Language,* 4a. ed. Boston: Kluwer Academic Publishers.

6 Registros y contadores

6-1 REGISTROS

Un circuito secuencial con reloj consiste en un grupo de flip-flops y compuertas combinacionales conectados para formar un camino de retroalimentación. Los flip-flops son indispensables porque, sin ellos, el circuito se reduce a un circuito puramente combinacional (suponiendo que no haya retroalimentación entre las compuertas). Un circuito con flip-flops se considera secuencial aunque no tenga compuertas combinacionales. Los circuitos que incluyen flip-flops por lo regular se clasifican según la función que desempeñan, más que por el nombre del circuito secuencial. Dos de esos circuitos son los registros y los contadores.

Un registro es un grupo de flip-flops. Cada flip-flop puede almacenar un bit de información. Un registro de n bits consiste en un grupo de n flip-flops capaces de almacenar n bits de información binaria. Además de los flip-flops, un registro puede tener compuertas combinacionales que realizan ciertas tareas de procesamiento de datos. En su definición más amplia, un registro consiste en un grupo de flip-flops y compuertas que efectúan su transición. Los flip-flops contienen la información binaria y las compuertas determinan cómo esa información se transfiere al registro.

Un contador es básicamente un registro que pasa por una sucesión predeterminada de estados. Las compuertas del contador están conectadas de tal manera que producen la sucesión prescrita de estados binarios. Aunque los contadores son un tipo especial de registros, es común distinguirlos dándoles otro nombre.

Hay diversos tipos de registros en el comercio. El más sencillo consiste únicamente en flip-flops, sin compuertas. La figura 6-1 muestra uno de esos registros construido con cuatro flip-flops tipo D. La entrada de reloj, común a todos los flip-flops, los dispara en el flanco positivo de cada pulso, y los datos binarios disponibles en las cuatro entradas se transfieren en el registro de cuatro bits. Es posible muestrear las cuatro salidas en cualquier momento para obtener la información binaria almacenada en el registro. La entrada de despeje (*clear*) se conecta a la entrada R (restablecimiento, *reset*) de los cuatro flip-flops. Cuando esta entrada cambia

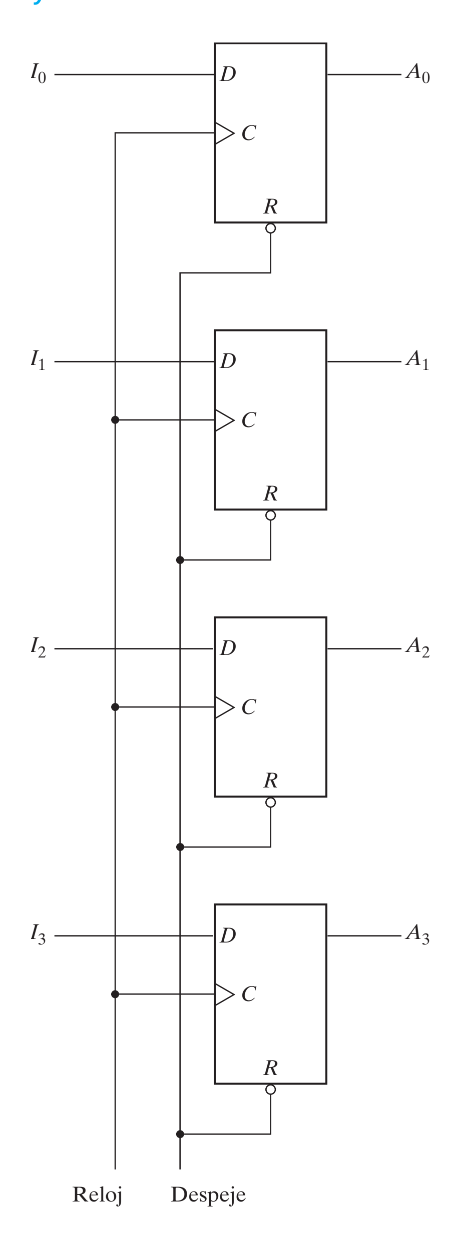


FIGURA 6-1
Registro de cuatro bits

a 0, todos los flip-flops se restablecen asincrónicamente. La entrada *clear* sirve para poner en ceros el registro antes de que comience a funcionar con reloj. Las entradas R se deben mantener en 1 lógico durante el funcionamiento normal con reloj. Se utiliza tanto *clear* como *reset* para indicar la transferencia del registro al estado de ceros.

Registro con carga paralela

Los sistemas digitales sincrónicos tienen un generador maestro de reloj que suministra un tren continuo de pulsos de reloj. Estos pulsos se aplican a todos los flip-flops y registros del sistema. El reloj maestro actúa como una bomba que alimenta un latido constante a todas las partes del sistema. Se requiere una señal de control aparte para decidir qué pulso de reloj específico tendrá efecto sobre un registro dado. La transferencia de información nueva a un registro se describe como *carga* del registro. Si todos los bits del registro se cargan simultáneamente, con un pulso de reloj común, se dice que la carga se efectúa en paralelo. Un borde de reloj aplicado a las entradas *C* del registro de la figura 6-1 carga las cuatro entradas en paralelo. En esta configuración, el reloj deberá inhibirse del circuito cuando se desee que el registro conserve intacto su contenido. Esto se hace controlando la señal de entrada del reloj con una compuerta habilitadora. Sin embargo, la inserción de compuertas en la trayectoria del reloj implica que la lógica se efectúa con pulsos de reloj. La inserción de compuertas lógicas produce retardos de propagación desiguales entre el reloj maestro y las entradas de los flip-flops. Para sincronizar plenamente el sistema, hay que cerciorarse de que todos los pulsos de reloj lleguen al mismo tiempo a todos los puntos del sistema, para que todos los flip-flops se disparen en forma simultánea. La lógica efectuada con pulsos de reloj inserta retardos variables y podría hacer que el sistema se desincronice. Por ello, es aconsejable controlar el funcionamiento del registro con las entradas *D*, en vez de controlar el reloj en las entradas *C* de los flip-flops.

En la figura 6-2 se observa un registro de cuatro bits con una entrada de control de carga que se hace pasar por compuertas y llega a las entradas *D* de los flip-flops. La entrada de carga del registro determina la acción que se realizará en cada pulso de reloj. Si la entrada de carga es 1, los datos de las cuatro entradas se transfieren al registro en el siguiente borde positivo del reloj. Si la entrada de carga es 0, las salidas de los flip-flops se conectan a sus respectivas entradas. La conexión de retroalimentación de salida a entrada es necesaria porque el flip-flop *D* no tiene una condición de “sin cambio”. En cada flanco de reloj, la entrada *D* determina el siguiente estado del registro. Para que la salida no cambie, es necesario hacer que la entrada *D* sea igual al valor actual de la salida.

Los pulsos de reloj se aplican a las entradas *C* en todo momento. La entrada de carga determina si el siguiente pulso va a aceptar nueva información o va a dejar intacta la información que está en el registro. La transferencia de información de las entradas de datos o las salidas del registro se efectúa simultáneamente con los cuatro bits, en respuesta a un borde de reloj.

6-2 REGISTROS DE DESPLAZAMIENTO

Un registro capaz de desplazar su información binaria en una dirección o en la otra se llama *registro de desplazamiento*. La configuración lógica de un registro de desplazamiento consiste en una cadena de flip-flops en cascada, con la salida de un flip-flop conectada a la entrada del siguiente flip-flop. Todos los flip-flops reciben pulsos de reloj comunes, que activan el desplazamiento de una etapa a la siguiente.

El registro de desplazamiento más sencillo posible usa sólo flip-flops, como se indica en la figura 6-3. La salida de un flip-flop dado se conecta a la entrada *D* del flip-flop que está a su derecha. Cada pulso de reloj desplaza el contenido del registro una posición de bit a la derecha. La *entrada en serie* determina qué sucede en el flip-flop de la extrema izquierda durante el desplazamiento. La *salida en serie* se toma de la salida del flip-flop de la extrema derecha. A veces se hace necesario controlar el desplazamiento de modo que sólo se efectúe con ciertos pulsos, pe-

FIGURA 6-2
Registro de cuatro bits con carga paralela

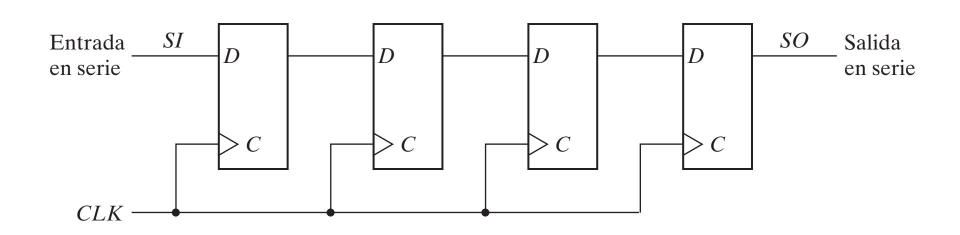


FIGURA 6-3
Registro de desplazamiento de cuatro bits

ro no con otros. Esto se logra inhibiendo el reloj de la entrada del registro para impedir que se desplace. Más adelante se verá que la operación de desplazamiento se controla a través de las entradas *D* de los flip-flops, en vez de hacerse a través de la entrada de reloj. Pero si se se usa el registro de desplazamiento de la figura 6-3, el desplazamiento podrá controlarse conectando el reloj a través de una compuerta AND con una entrada que controle el desplazamiento.

Transferencia en serie

Decimos que un sistema digital opera en modo en serie cuando la información se transfiere y manipula bit por bit. La información se transfiere bit por bit desplazando los bits del registro de origen hacia el registro de destino. Esto contrasta con la transferencia paralela, en la que todos los bits del registro se transfieren al mismo tiempo.

La transferencia en serie de información del registro *A* al registro *B* se efectúa con registros de desplazamiento, como se observa en el diagrama de bloques de la figura 6-4a). La salida en serie (*SO*) del registro *A* se conecta a la entrada en serie (*SI*) del registro *B*. Para evitar la pérdida de información almacenada en el registro de origen, se hace que la información del registro *A* circule conectando la salida en serie con la entrada en serie. El contenido inicial del registro *B* se desplaza hacia su salida en serie y se pierde a menos que se transfiera a un tercer registro de desplazamiento. La entrada de control de desplazamiento determina cuándo y cúantas veces se desplazan los registros. Esto se hace con una compuerta AND que permite el paso de pulsos de reloj a las terminales *CLK* únicamente cuando el control de desplazamiento está activo.

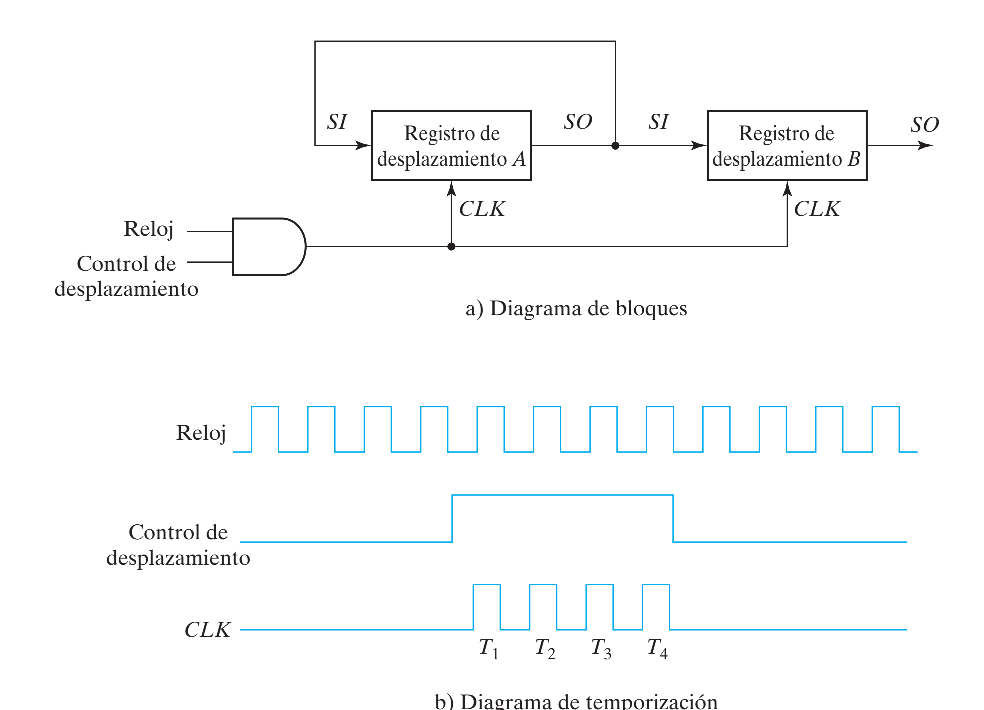


FIGURA 6-4
Transferencia en serie del registro *A* al registro *B*

Tabla 6-1
Ejemplo de transferencia en serie

Pulso de temporización	Registro de desplazamiento *A*	Registro de desplazamiento *B*
Valor inicial	1 0 1 1	0 0 1 0
Después de T_1	1 1 0 1	1 0 0 1
Después de T_2	1 1 1 0	1 1 0 0
Después de T_3	0 1 1 1	0 1 1 0
Después de T_4	1 0 1 1	1 0 1 1

Supongamos que cada uno de los registros de desplazamiento tiene cuatro bits. La unidad de control que supervisa la transferencia debe diseñarse de tal manera que habilite los registros de desplazamiento, a través de la señal de control de desplazamiento, durante un tiempo fijo de cuatro pulsos de reloj. Esto se muestra en el diagrama de temporización de la figura 6-4b). La señal de control de desplazamiento se sincroniza con el reloj y cambia de valor inmediatamente después del flanco negativo del reloj. Los cuatro pulsos de reloj siguientes encuentran la señal de control de desplazamiento en el estado activo, así que la salida de la compuerta AND conectada a las entradas *CLK* produce cuatro pulsos, T_1, T_2, T_3 y T_4. Cada borde ascendente del pulso causa un desplazamiento en ambos registros. El cuarto pulso cambia el control de desplazamiento a 0 y los registros de desplazamiento quedan inhabilitados.

Supongamos que el contenido binario de *A* antes del desplazamiento es 1011 y que el de *B* es 0010. La transferencia en serie de *A* a *B* se efectúa en cuatro pasos, como se indica en la tabla 6-1. Con el primer pulso T_1, el bit de la extrema derecha de *A* se desplaza al bit de la extrema izquierda de *B* y también se circula a la posición de extrema izquierda de *A*. Al mismo tiempo, todos los bits de *A* y *B* se desplazan una posición a la derecha. La anterior salida en serie de *B*, en la posición de extrema derecha, se pierde, y su valor cambia de 0 a 1. Los tres pulsos siguientes efectúan operaciones idénticas, desplazando los bits de *A* a *B*, uno por uno. Después del cuarto desplazamiento, el control de desplazamiento cambia a 0 y ambos registros, *A* y *B*, tienen el valor 1011. Así, el contenido de *A* se transfiere a *B*, pero permanece inalterado.

La diferencia entre los modos de operación en serie y en paralelo deberá ser obvia por este ejemplo. En el modo paralelo, se cuenta con información de todos los bits de un registro, y todos los bits se pueden transferir simultáneamente durante un pulso de reloj. En el modo en serie, los registros tienen una sola entrada en serie y una sola salida en serie. La información se transfiere bit por bit mientras los registros se desplazan en la misma dirección.

Suma en serie

Las operaciones de las computadoras digitales por lo regular se efectúan en paralelo porque este modo de operación es más rápido. Las operaciones en serie son más lentas, pero tienen la ventaja de requerir menos equipo. Para ilustrar el modo de operación en serie, presentaremos aquí el diseño de un sumador en serie. Su contraparte paralela se presentó en la sección 4-4.

Los dos números binarios que se sumarán en serie se almacenan en dos registros de desplazamiento. Los bits se suman par por par utilizando un solo circuito de sumador completo (SC), como se observa en la figura 6-5. El acarreo de salida del sumador completo se transfiere a un flip-flop *D*. La salida de este flip-flop se utiliza entonces como acarreo de entrada para el siguiente par de bits significativos. El bit de suma de la salida *S* del sumador completo podría transferirse a un tercer registro de desplazamiento. Al desplazar la suma a *A* mientras se des-

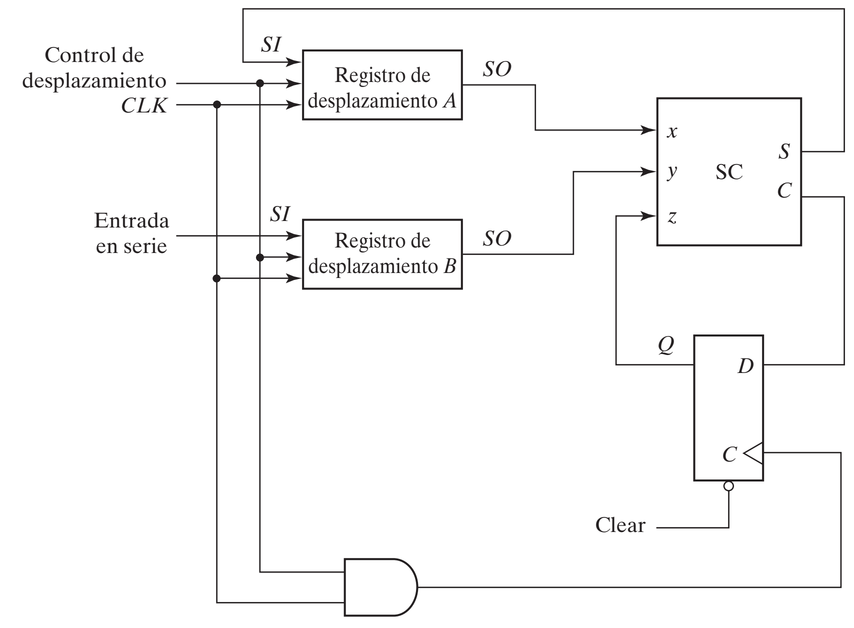


FIGURA 6-5
Sumador en serie

plazan hacia afuera los bits de *A*, es posible utilizar un solo registro para almacenar los bits tanto de un sumando como de la suma. La entrada en serie del registro *B* sirve para transferir a *B* un nuevo número binario mientras los bits del sumando se desplazan hacia afuera durante la suma.

El sumador en serie funciona como sigue. Inicialmente, el registro *A* contiene el primer sumando, el registro *B* contiene el segundo sumando y el flip-flop de acarreo está en 0. Las salidas (*SO*) de *A* y *B* alimentan un par de bits significativos al sumador completo en *x* y *y*. La salida *Q* del flip-flop alimenta el acarreo de entrada en *z*. El control de desplazamiento habilita ambos registros y el flip-flop de acarreo, de modo que, en el siguiente pulso de reloj, ambos registros se desplazarán una vez a la derecha; el bit de suma de *S* ingresará en el flip-flop de extrema izquierda de *A*, y el acarreo de salida se transferirá al flip-flop *Q*. El control de desplazamiento habilita los registros durante un número de pulsos de reloj igual al número de bits que hay en los registros. Con cada pulso de reloj sucesivo, un nuevo bit de suma se transfiere a *A*, un nuevo acarreo se transfiere a *Q* y ambos registros se desplazan una vez a la derecha. Este proceso continúa hasta que el control de desplazamiento se inhabilita. Así pues, la suma se efectúa pasando cada par de bits, junto con el acarreo anterior, por un solo circuito de sumador completo, y transfiriendo la suma, bit por bit, al registro *A*.

Inicialmente, *A* y el flip-flop de acarreo se ponen en 0 (con *clear*), y luego se suma el primer número de *B*. Mientras *B* se desplaza a través del sumador completo, se transfiere a él un segundo número a través de su entrada en serie. Luego el segundo número se suma al contenido del registro *A* mientras un tercer número se transfiere en serie al registro *B*. Esto puede repetirse para efectuar la suma de dos, tres o más números y acumular su suma en el registro *A*.

Si se compara el sumador en serie con el sumador paralelo descrito en la sección 4-4, notaremos varias diferencias. El sumador paralelo utiliza registros de carga paralela, mientras que el sumador en serie usa registros de desplazamiento. El número de circuitos de sumador

completo en el sumador paralelo es igual al número de bits de los números binarios, mientras que el sumador en serie sólo requiere un circuito de sumador completo y un flip-flop de acarreo. Sin contar los registros, el sumador paralelo es un circuito combinacional, mientras que el sumador en serie es un circuito secuencial. El circuito secuencial del sumador en serie consiste en un sumador completo y un flip-flop que almacena el acarreo de salida. Esto es típico en las operaciones en serie porque el resultado de una operación en un tiempo de un bit podría depender no sólo de las entradas actuales, sino también de las entradas anteriores que se deben almacenar en flip-flops.

Para demostrar que es posible diseñar operaciones en serie empleando el procedimiento de circuitos secuenciales, volveremos a diseñar el sumador en serie utilizando una tabla de estados. Primero, suponga que contamos con dos registros de desplazamiento para almacenar los números binarios que se sumarán en serie. Las salidas en serie de los registros se designarán x y y. El circuito secuencial a diseñar no incluirá los registros de desplazamiento; éstos se insertarán después para mostrar el circuito completo. El circuito secuencial propiamente dicho tiene las dos entradas, x y y, que alimentan un par de bits significativos, una salida S que genera el bit de suma y un flip-flop Q para almacenar el acarreo. La tabla de estados que especifica el circuito secuencial se presenta en la tabla 6-2. El estado actual de Q es el valor actual del acarreo. El acarreo que está en Q se suma a las entradas x y y para producir el bit de suma en la salida S. El siguiente estado de Q es igual al acarreo de salida. Vemos que las filas de la tabla de estados son idénticas a las de la tabla de verdad de un sumador completo, excepto que el acarreo de entrada ahora es el estado actual de Q y el acarreo de salida ahora es el siguiente estado de Q.

Si usamos un flip-flop D para Q, el circuito se reducirá al que se muestra en la figura 6-5. Si usamos un flip-flop JK para Q, será necesario determinar los valores de las entradas J y K consultando la tabla de excitación (tabla 5-12). Esto se hace en las últimas dos columnas de la tabla 6-2. Las dos ecuaciones de entrada de flip-flop y la ecuación de salida se simplifican por medio de mapas para obtener

$$J_Q = xy$$
$$K_Q = x'y' = (x + y)'$$
$$S = x \oplus y \oplus Q$$

Tabla 6-2
Tabla de estados del sumador en serie

Estado actual	Entradas		Siguiente estado	Salida	Entradas del flip-flop	
Q	x	y	Q	S	J_Q	K_Q
0	0	0	0	0	0	X
0	0	1	0	1	0	X
0	1	0	0	1	0	X
0	1	1	1	0	1	X
1	0	0	0	1	X	1
1	0	1	1	0	X	0
1	1	0	1	0	X	0
1	1	1	1	1	X	0

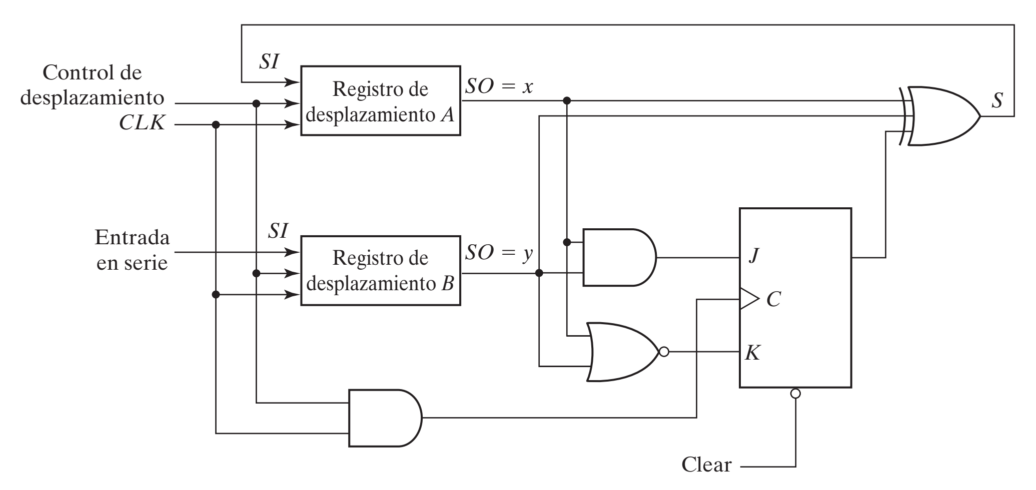


FIGURA 6-6
Segunda forma del sumador en serie

El diagrama de circuito se reproduce en la figura 6-6. El circuito consta de tres compuertas y un flip-flop *JK*. Se han incluido los dos registros de desplazamiento en el diagrama para mostrar el sumador en serie completo. Cabe señalar que la salida *S* es función no sólo de *x* y *y*, sino también del estado actual de *Q*. El siguiente estado de *Q* es función del estado actual de *Q* y de los valores *x* y *y* que salen de las salidas en serie de los registros de desplazamiento.

Registro de desplazamiento universal

Si las salidas de flip-flop de un registro de desplazamiento están accesibles, la información que se introduce en serie por desplazamiento se puede sacar en paralelo de las salidas de los flip-flops. Si se añade la capacidad de carga en paralelo a un registro de desplazamiento, los datos introducidos en paralelo se podrán sacar en serie desplazando los datos almacenados en el registro.

Algunos registros de desplazamiento proporcionan las terminales de entrada y salida necesarias para la transferencia en paralelo. También podrían tener capacidad de desplazamiento tanto a la derecha como a la izquierda. El registro de desplazamiento más general tiene estas capacidades:

1. Un control de despeje (*clear*) para poner en ceros el registro.
2. Una entrada de reloj (*clock*) para sincronizar las operaciones.
3. Un control de desplazamiento a derecha (*shift-right*) para habilitar la operación de desplazamiento a la derecha, y las líneas de *entrada* y *salida en serie* asociadas al desplazamiento a la derecha.
4. Un control de desplazamiento a izquierda (*shift-left*) para habilitar la operación de desplazamiento a la izquierda, y las líneas de *entrada* y *salida en serie* asociadas al desplazamiento a la izquierda.
5. Un control de carga en paralelo (*parallel-load*) para habilitar la transferencia en paralelo y las *n* líneas de entrada asociadas a la transferencia en paralelo.
6. *n* líneas de salida en paralelo.
7. Un control de estado que deja la información del registro como está, en presencia del reloj.

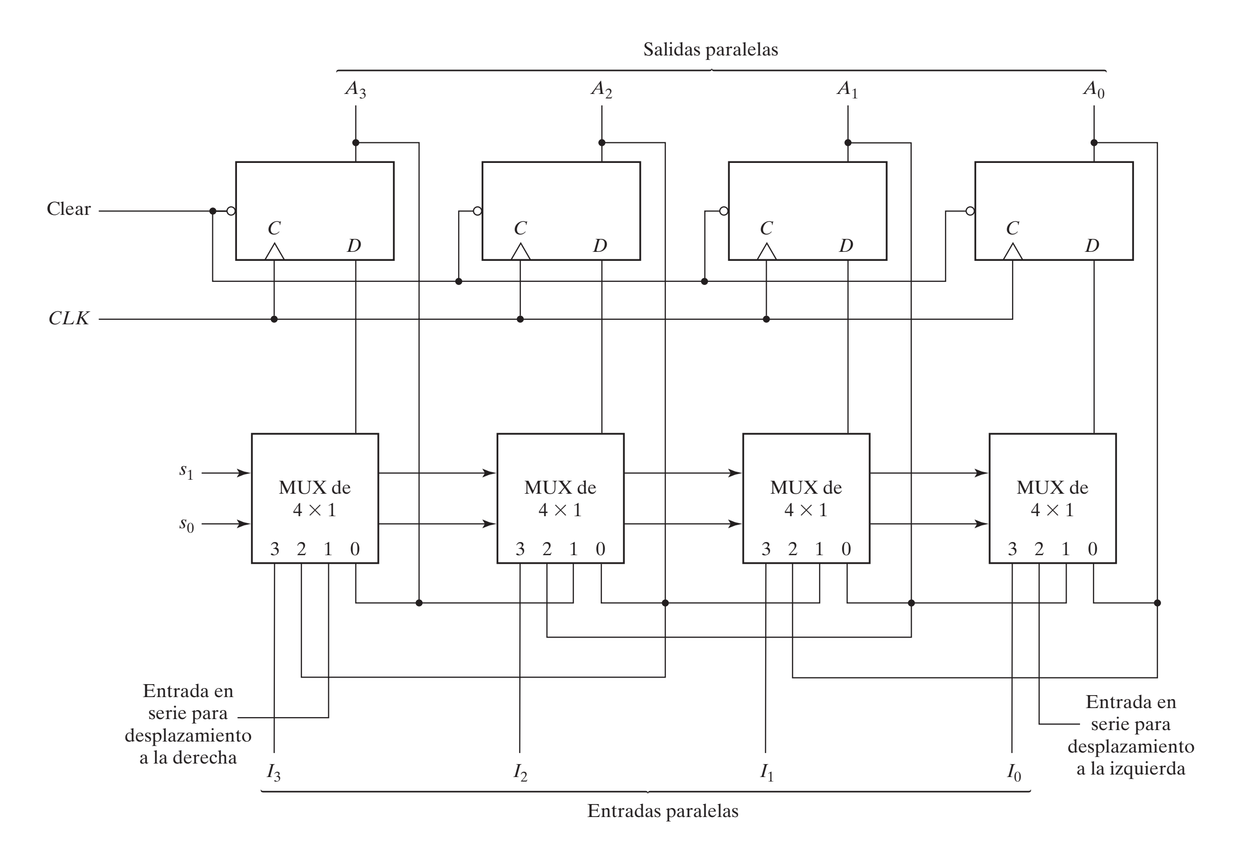


FIGURA 6-7
Registro de desplazamiento universal de 4 bits

Otros registros de desplazamiento podrían tener sólo algunas de las funciones anteriores, con una operación de desplazamiento por lo menos.

Un registro que sólo puede desplazar en una dirección es un registro de desplazamiento unidireccional. Uno que puede hacerlo en ambas direcciones es un registro de desplazamiento bidireccional. Si el registro tiene ambos desplazamientos y capacidad de carga paralela, se denomina *registro de desplazamiento universal*.

En la figura 6-7 se presenta el diagrama de un registro de desplazamiento universal de cuatro bits que posee todas las capacidades de la lista anterior. Consta de cuatro flip-flops *D* y cuatro multiplexores. Los cuatro multiplexores tienen dos entradas de selección en común, s_1 y s_0. La entrada 0 de cada multiplexor se selecciona cuando $s_1s_0 = 00$, la entrada 1 se selecciona cuando $s_1s_0 = 01$, y de manera análoga para las otras dos entradas. Las entradas de selección controlan el modo de operación del registro según las funciones enumeradas en la tabla 6-3. Cuando $s_1s_0 = 00$, el valor actual del registro se aplica a las entradas *D* de los flip-flops. Esta condición establece una trayectoria desde la salida de cada flip-flop hasta la entrada del mismo flip-flop. El siguiente borde de reloj transfiere a cada flip-flop el valor binario que con-

Tabla 6-3
Tabla de función para el registro de la figura 6-7

Control de modo		
s_1	s_0	**Operación del *registro***
0	0	Sin cambio
0	1	Desplazamiento a la derecha
1	0	Desplazamiento a la izquierda
1	1	Carga en paralelo

tenía antes, así que no hay cambio de estado. Cuando $s_1s_0 = 01$, la terminal 1 de las entradas de multiplexor tiene una trayectoria a las entradas D de los flip-flops. Esto causa una operación de desplazamiento a la derecha, transfiriéndose la entrada en serie al flip-flop A_3. Cuando $s_1s_0 = 10$, el resultado es una operación de desplazamiento a la izquierda, y la otra entrada en serie pasa al flip-flop A_0. Por último, cuando $s_1s_0 = 11$, la información binaria que está en las líneas de entrada paralelas se transfiere al registro simultáneamente durante el siguiente borde de reloj.

Los registros de desplazamiento se usan mucho como interfaz de sistemas digitales situados lejos uno del otro. Por ejemplo, supongamos que es necesario transmitir una cantidad de n bits entre dos puntos. Si la distancia es grande, sería costoso usar n líneas para transmitir los n bits en paralelo. Resulta más económico usar una sola línea y transmitir la información en serie, bit por bit. Los n bits de datos se colocan en paralelo en un registro del transmisor y luego se transmiten en serie por la línea común. El receptor acepta los datos en serie en un registro de desplazamiento. Una vez que ha recibido los n bits, éstos se pueden tomar de las salidas del registro en paralelo. Así, el transmisor efectúa una conversión de los datos, de paralelo a serie, y el receptor efectúa una conversión de serie a paralelo.

6-3 CONTADORES DE RIZO

Un registro que pasa por una sucesión preescrita de estados cuando se aplican pulsos de entrada se denomina contador. Los pulsos de entrada podrían ser pulsos de reloj u originarse en alguna fuente externa, y podrían presentarse a intervalos fijos de tiempo o al azar. La sucesión de estados podría seguir la sucesión numérica binaria o cualquier otro orden. Un contador que sigue la sucesión numérica binaria es un contador binario. Un contador binario de n bits consiste en n flip-flops y puede contar en binario desde 0 hasta $2^n - 1$.

Los contadores se dividen en dos categorías: contadores de rizo y contadores sincrónicos. En un contador de rizo, la transición de salida del flip-flop sirve como disparador de otros flip-flops. Dicho de otro modo, la entrada C de algunos flip-flops, o de todos, se dispara, no con los pulsos del reloj común, sino con la transición que se da en otras salidas de flip-flop. En un contador sincrónico, las salidas C de todos los flip-flops reciben el reloj común. Se hablará de los contadores sincrónicos en las dos secciones siguientes. Aquí presentaremos los contadores de rizo binario y BCD y explicaremos su funcionamiento.

Contador binario de rizo

Un contador binario de rizo consiste en una conexión en serie de flip-flops complementadores; la salida de cada flip-flop se conecta a la entrada *C* del siguiente flip-flop de orden superior. El flip-flop que contiene el bit menos significativo recibe los pulsos de conteo que llegan. Es posible construir un flip-flop complementador con un flip-flop *JK* cuyas entradas *J* y *K* se han conectado entre sí, o con un flip-flop *T*. Una tercera posibilidad es usar un flip-flop *D* con la salida de complemento conectada a la entrada *D*. Así, la entrada *D* siempre es el complemento del estado actual, y el siguiente pulso de reloj hace que el flip-flop se complemente. En la figura 6-8 se presenta el diagrama lógico de dos contadores binarios de rizo de cuatro bits. El contador se construye con flip-flops complementadores del tipo *T* en la parte a) y de tipo *D* en la parte b). La salida de cada flip-flop se conecta a la entrada *C* del siguiente flip-flop sucesivo. El flip-flop que contiene el bit menos significativo recibe los pulsos de conteo que llegan. Las entradas *T* de todos los flip-flops de a) se conectan a un 1 lógico permanente. Esto hace que cada flip-flop se complemente si la señal de su entrada *C* sufre una transición negativa. La burbuja junto al símbolo de indicador dinámico de *C* denota que los flip-flops responden a la transición de borde negativo de la entrada. La transición negativa se presenta cuando la salida del flip-flop anterior al que *C* está conectada cambia de 1 a 0.

Para entender el funcionamiento del contador binario de rizo de cuatro bits, resulta útil remitirse a la lista de los primeros nueve números binarios de la tabla 6-4. El conteo inicia con el 0 binario y se incrementa en uno con cada pulso de conteo introducido. Después de la cuenta de 15, el contador vuelve a 0 para repetir la cuenta. El bit menos significativo, A_0, se complementa con cada pulso de conteo introducido. Cada vez que A_0 pasa de 1 a 0, complementa a A_1. Cada vez que A_1 pasa de 1 a 0, complementa a A_2. Cada vez que A_2 pasa de 1 a 0, complementa a A_3, y así sucesivamente con los demás bits de orden más alto que tenga el contador. Por ejemplo, considere la transición de la cuenta 0011 a 0100. A_0 se complementa con el pulso de conteo. Puesto que A_0 pasa de 1 a 0, dispara a A_1 y lo complementa. El resultado de esto es que A_1 cambia de 1 a 0, lo que a su vez hace que A_2 se complemente, cambiando de 0 a 1. A_2 no dispara a A_3 porque A_2 produce una transición positiva y el flip-flop sólo responde a transiciones negativas. Así, el conteo de 0011 a 0100 se logra cambiando los bits uno por uno, de modo que el conteo pasa primero de 0011 a 0010, luego a 0000 y por último a 0100. Los flip-flops cambian uno por uno sucesivamente, y la señal se propaga por el contador como rizo, de una etapa a la siguiente.

Un contador binario que cuenta al revés se llama contador binario de cuenta regresiva. En él, la cuenta binaria se decrementa en uno con cada pulso de conteo que llega. La cuenta de un contador de cuenta regresiva de cuatro bits inicia en el 15 binario y continúa con las cuentas binarias 14, 13, 12, . . . , 0 y luego de vuelta a 15. Una lista de la sucesión de conteo de un contador binario de cuenta regresiva muestra que el bit menos significativo se complementa con cada pulso de conteo. Cualquier otro bit de la sucesión se complementa si el bit menos significativo precedente pasa de 0 a 1. Por tanto, el diagrama de un contador binario de cuenta regresiva es igual al de la figura 6-8, a condición de que todos los flip-flops se disparen con el borde positivo del reloj. (No debe haber burbuja en las entradas *C*.) Si se usan flip-flops disparados por borde negativo, la entrada *C* de cada flip-flop deberá conectarse a la salida de complemento del flip-flop anterior. Así, cuando la salida verdadera cambie de 0 a 1, el complemento cambiará de 1 a 0 y complementará el siguiente flip-flop, como debe ser.

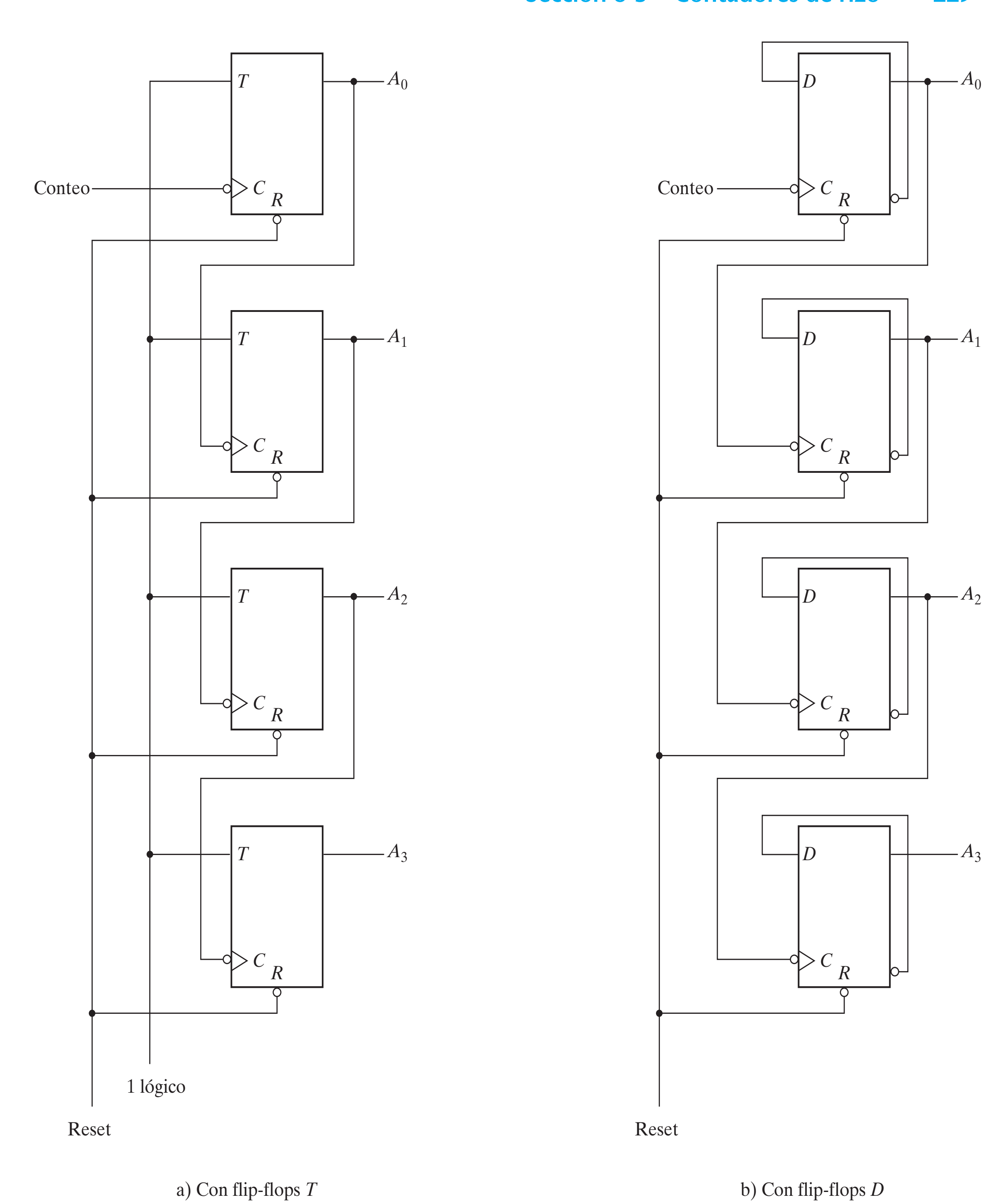


FIGURA 6-8
Contador binario de rizo de cuatro bits

Tabla 6-4
Sucesión binaria de conteo

A_3	A_2	A_1	A_0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0

Contador BCD de rizo

Un contador decimal sigue una sucesión de diez estados y vuelve a 0 después de la cuenta de 9. Un contador así necesita por lo menos cuatro flip-flops para representar cada dígito decimal, ya que un dígito decimal se representa con un código binario de por lo menos cuatro bits. La sucesión de estados de un contador decimal depende del código binario empleado para representar un dígito decimal. Si se usa BCD, la sucesión de estados es la que se aprecia en el diagrama de estados de la figura 6-9. Este contador es similar al binario, excepto que el estado que sigue a 1001 (código del dígito decimal 9) es 0000 (código para el dígito decimal 0).

En la figura 6-10 se presenta el diagrama lógico de un contador BCD de rizo que utiliza flip-flops *JK*. Las cuatro salidas se designan con la letra *Q* seguida de un subíndice numérico igual al peso binario del bit correspondiente en el código BCD. Vemos que la salida de Q_1 se aplica a las entradas *C* tanto de Q_2 como de Q_8, y que la salida de Q_2 se aplica a la entrada *C* de Q_4. Las entradas *J* y *K* se conectan a una señal de 1 lógico permanente o bien a salidas de otros flip-flops.

Un contador de rizo es un circuito secuencial asincrónico. Las señales que afectan la transición del flip-flop dependen de la forma en que cambian de 1 a 0. Podemos explicar el funcionamiento del contador con una lista de condiciones para transiciones de flip-flop. Estas condiciones se deducen del diagrama lógico y del conocimiento de la forma en que opera un flip-flop *JK*. Recuerde que, cuando la entrada *C* cambia de 1 a 0, el flip-flop se

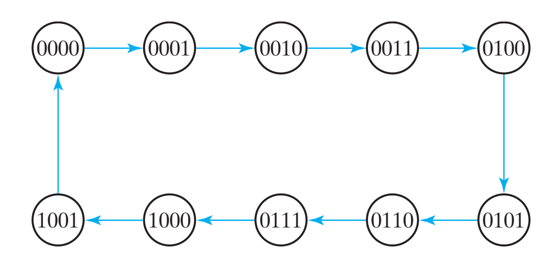


FIGURA 6-9
Diagrama de estados de un contador BCD decimal

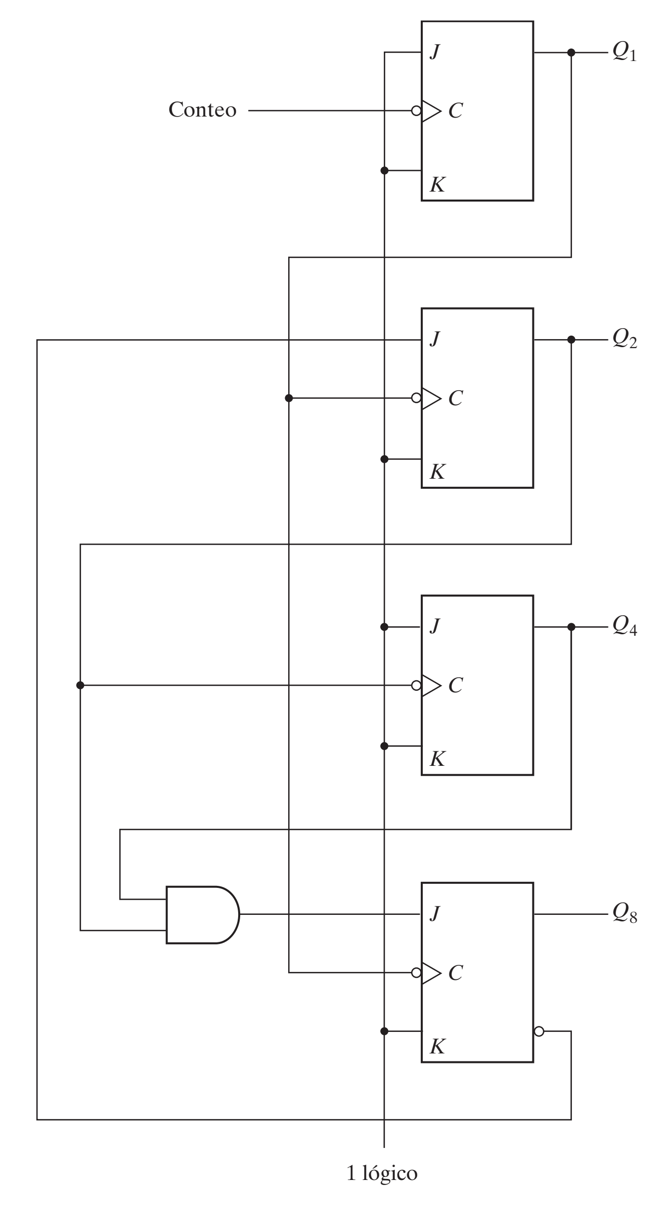


FIGURA 6-10
Contador BCD de rizo

establece si $J = 1$, se restablece si $K = 1$, se complementa si $J = K = 1$ y no cambia si $J = K = 0$.

Para verificar que estas condiciones producen la sucesión requerida por un contador BCD de rizo, es necesario comprobar que las transiciones de flip-flop pasen sucesivamente a los estados especificados por el diagrama de estados de la figura 6-9. Q_1 cambia de estado después de cada pulso de reloj. Q_2 se complementa cada vez que Q_1 pasa de 1 a 0, en tanto $Q_8 = 0$.

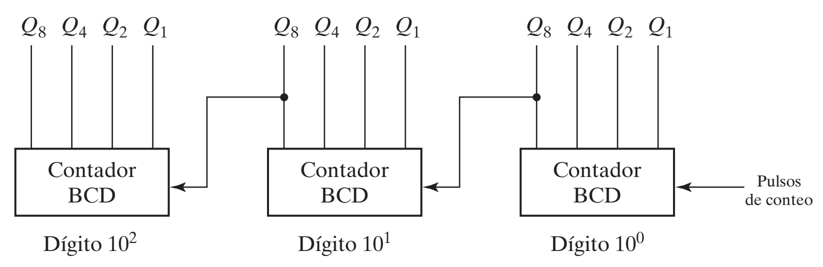


FIGURA 6-11
Diagrama de bloques de un contador BCD decimal de tres décadas

Cuando Q_8 cambia a 1, Q_2 permanece en 0. Q_4 se complementa cada vez que Q_2 cambia de 1 a 0. Q_8 permanece en 0 en tanto Q_2 o Q_4 sean 0. Si tanto Q_2 como Q_4 cambian a 1, Q_8 se complementa cuando Q_1 cambia de 1 a 0. Q_8 se despeja en la siguiente transición de Q_1.

El contador BCD de la figura 6-10 es un contador de *década*, pues cuenta de 0 a 9. Para contar en decimal de 0 a 99, se necesita un contador de dos décadas. Para contar de 0 a 999, es necesario uno de tres décadas. Los contadores de varias décadas se construyen conectando contadores BCD en cascada, uno para cada década. En la figura 6-11 se ilustra un contador de tres décadas. Las entradas de la segunda y la tercera décadas provienen de Q_8 de la década anterior. Cuando Q_8 de una década cambia de 1 a 0, dispara el conteo de la siguiente década de orden superior mientras su propia década cambia de 9 a 0.

6-4 CONTADORES SINCRÓNICOS

Los contadores sincrónicos difieren de los de rizo en que se aplican pulsos de reloj a las entradas de todos los flip-flops. Un mismo reloj dispara todos los flip-flops simultáneamente en vez de hacerlo uno por uno sucesivamente como en los contadores de rizo. La decisión de si un flip-flop debe complementarse o no depende de los valores de las entradas de datos, como T o J y K, en el momento en que llega el borde de reloj. Si $T = 0$ o $J = K = 0$, el flip-flop no cambia de estado. Si $T = 1$ o $J = K = 1$, el flip-flop se complementa.

Ya presentamos el procedimiento de diseño de los contadores sincrónicos en la sección 5-7, y el diseño de un contador binario de tres bits se efectuó con la ayuda de la figura 5-31. En esta sección se presentarán algunos contadores sincrónicos representativos y se explicará su funcionamiento.

Contador binario

El diseño de un contador binario sincrónico es tan sencillo que no es preciso realizar un proceso secuencial de diseño lógico. En un contador binario sincrónico, el flip-flop de la posición menos significativa se complementa con cada pulso. Un flip-flop en cualquier otra posición se complementa cuando todos los bits de las posiciones significativas inferiores son 1. Por ejemplo, si el estado actual de un contador de cuatro bits es $A_3A_2A_1A_0 = 0011$, el siguiente conteo será 0100. A_0 siempre se complementa. A_1 se complementa porque el estado actual de $A_0 = 1$. A_2 se complementa porque el estado actual de $A_1A_0 = 11$. En cambio, A_3 no se complementa porque el estado actual de $A_2A_1A_0 = 011$, y no cumple la condición de “todos unos”.

Los contadores binarios sincrónicos tienen un patrón regular y se pueden construir con flip-flops complementadores y compuertas. El patrón regular se distingue en el contador de cuatro bits que se representa en la figura 6-12. Las entradas C de todos los flip-flops se conectan a un reloj común. El contador se habilita con la entrada de habilitar contador. Si esa entrada es 0, todas las entradas J y K son 0 y el reloj no cambia el estado del contador. La primera etapa A_0

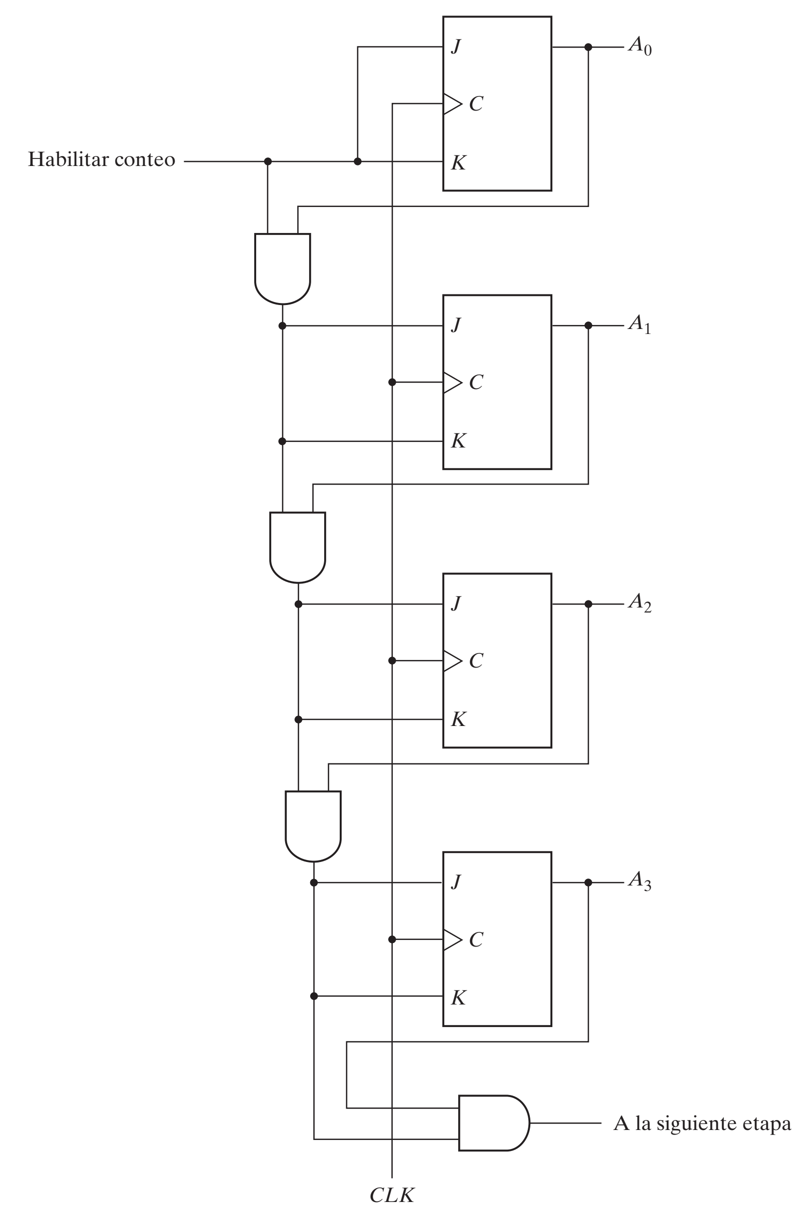


FIGURA 6-12
Contador binario sincrónico de cuatro bits

tiene 1 en su J y en su K si el contador está habilitado. Las otras entradas J y K tienen 1 si todas las etapas anteriores, menos significativas, producen 1 y el conteo está habilitado. La cadena de compuertas AND genera la lógica requerida para las entradas J y K de cada etapa. El contador puede extenderse a cualquier cantidad de etapas, cada una de las cuales tiene un flip-flop adicional y una compuerta AND que produce una salida de 1 si las salidas de todos los flip-flops anteriores son 1.

Cabe señalar que los flip-flops se disparan con el borde positivo del reloj. La polaridad del reloj no es fundamental en este caso como lo era en el contador de rizo. El contador sincrónico se dispara con el borde positivo del reloj, o con el negativo. Los flip-flops complementadores del contador binario pueden ser del tipo JK o del tipo T, o del tipo D con compuertas XOR. La equivalencia de los tres tipos se señala en la figura 5-13.

Contador binario ascendente-descendente

Un contador binario sincrónico de cuenta regresiva pasa por los estados binarios en el orden inverso, de 1111 hasta 0000, pasando después a 1111 para repetir el conteo. Es posible diseñar un contador de cuenta regresiva de la forma acostumbrada, pero el resultado es predecible por inspección del conteo binario descendente. El bit de la posición menos significativa se complementa con cada pulso. Un bit en cualquier otra posición se complementa si todos los bits menos significativos son 0. Por ejemplo, el estado que sigue al estado actual 0100 es 0011. El bit menos significativo siempre se complementa. El segundo bit hacia la izquierda se complementa porque el primero es 0. El tercero se complementa porque los dos primeros son 0. El cuarto bit, en contraste, no cambia porque no todos los bits menos significativos son 0.

Podemos construir un contador binario de cuenta regresiva como el de la figura 6-12, excepto que las entradas de las compuertas AND deben provenir de las salidas complementadas de los flip-flops anteriores, no de las salidas normales. Es posible combinar las dos operaciones en un solo circuito para formar un contador capaz de contar hacia arriba o hacia abajo. En la figura 6-13 se representa el circuito de un contador binario **ascendente-descendente** que utiliza flip-flops T. Tiene una entrada de control para conteo ascendente (arriba) y una entrada de control para conteo descendente (abajo). Cuando la entrada arriba es 1, el circuito cuenta hacia arriba, porque las entradas T reciben sus señales de las salidas normales de los flip-flops anteriores. Cuando la entrada abajo es 1 y la entrada arriba es 0, el circuito cuenta hacia abajo, porque se aplican a las entradas T las salidas complementadas de los flip-flops anteriores. Si ambas entradas, arriba y abajo, son 0, el circuito no cambia de estado y permanece en la misma cuenta. Si ambas entradas son 1, el circuito cuenta hacia arriba. Esto garantiza que sólo una operación se efectúe en todo momento.

Contador BCD

Los contadores BCD cuentan en decimal codificado en binario, de 0000 hasta 1001 y luego regresan a 0000. Debido al regreso a cero después de contar hasta 9, el contador BCD no sigue un patrón regular como en el conteo binario directo. Para deducir el circuito de un contador BCD sincrónico, es preciso efectuar un procedimiento secuencial de diseño de circuitos.

La tabla de estados de un contador BCD se presenta en la tabla 6-5. Las condiciones de entrada de los siete flip-flops se obtienen de las condiciones de estado actual y siguiente estado. También se da una salida y en la tabla. Esta salida es 1 cuando el estado actual es 1001. Así, y

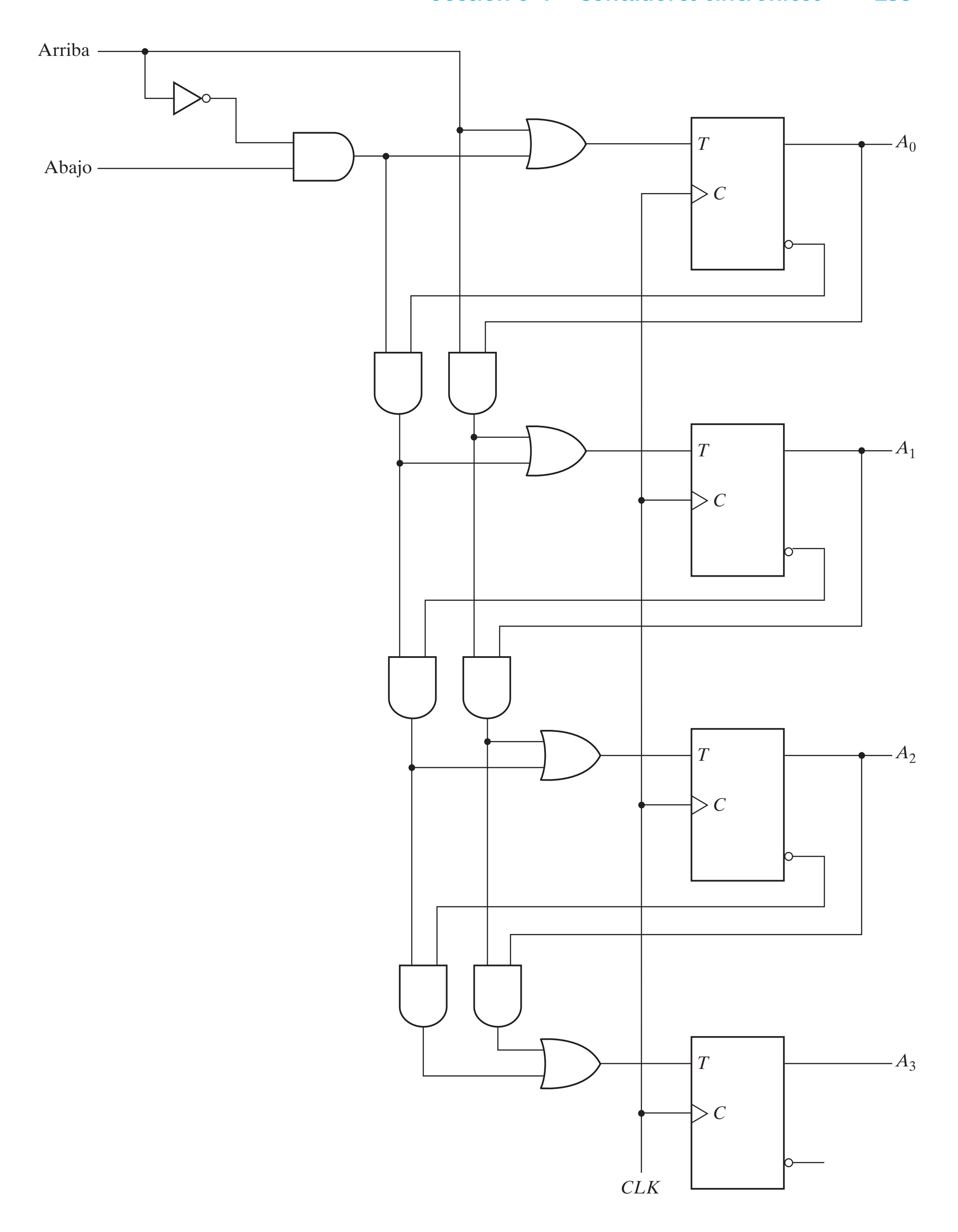


FIGURA 6-13
Contador binario ascendente-descendente de cuatro bits

Tabla 6-5
Tabla de estados para el contador BCD

Estado actual				Siguiente estado				Salida	Entradas de flip-flop			
Q_8	Q_4	Q_2	Q_1	Q_8	Q_4	Q_2	Q_1	y	TQ_8	TQ_4	TQ_2	TQ_1
0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	1	0	0	1	0	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	0	1
0	0	1	1	0	1	0	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	0	1
0	1	0	1	0	1	1	0	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	0	1
0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	0	0	1	0	0	0	0	1
1	0	0	1	0	0	0	0	1	1	0	0	1

puede habilitar el conteo de la siguiente década más significativa al tiempo que cambia la década actual de 1001 a 0000.

Las ecuaciones de entrada de los flip-flops se simplifican con la ayuda de mapas. Los estados no utilizados de los minitérminos 10 a 15 se toman como términos de indiferencia. Las funciones simplificadas son

$$T_{Q1} = 1$$
$$T_{Q2} = Q_8'Q_1$$
$$T_{Q4} = Q_2Q_1$$
$$T_{Q8} = Q_8Q_1 + Q_4Q_2Q_1$$
$$y = Q_8Q_1$$

Es fácil dibujar el circuito con cuatro flip-flops T, cinco compuertas AND y una compuerta OR. Los contadores BCD sincrónicos se pueden conectar en cascada para formar un contador de números decimales de cualquier longitud. La conexión en cascada se hace como en la figura 6-11, excepto que la salida y se debe conectar a la entrada de conteo de la siguiente década más significativa.

Contador binario con carga paralela

Es muy común que los contadores empleados en sistemas digitales requieran una capacidad de carga paralela para transferir un número binario inicial al contador antes de la operación de conteo. La figura 6-14 representa el diagrama lógico de un registro de cuatro bits que tiene capacidad de carga paralela y puede operar como contador. Si la entrada de control de carga es 1, la operación de conteo se inhabilita y se efectúa una transferencia de datos de las cuatro entradas de datos a los cuatro flip-flops. Si ambas entradas de control son 0, los pulsos de reloj no alteran el estado del registro.

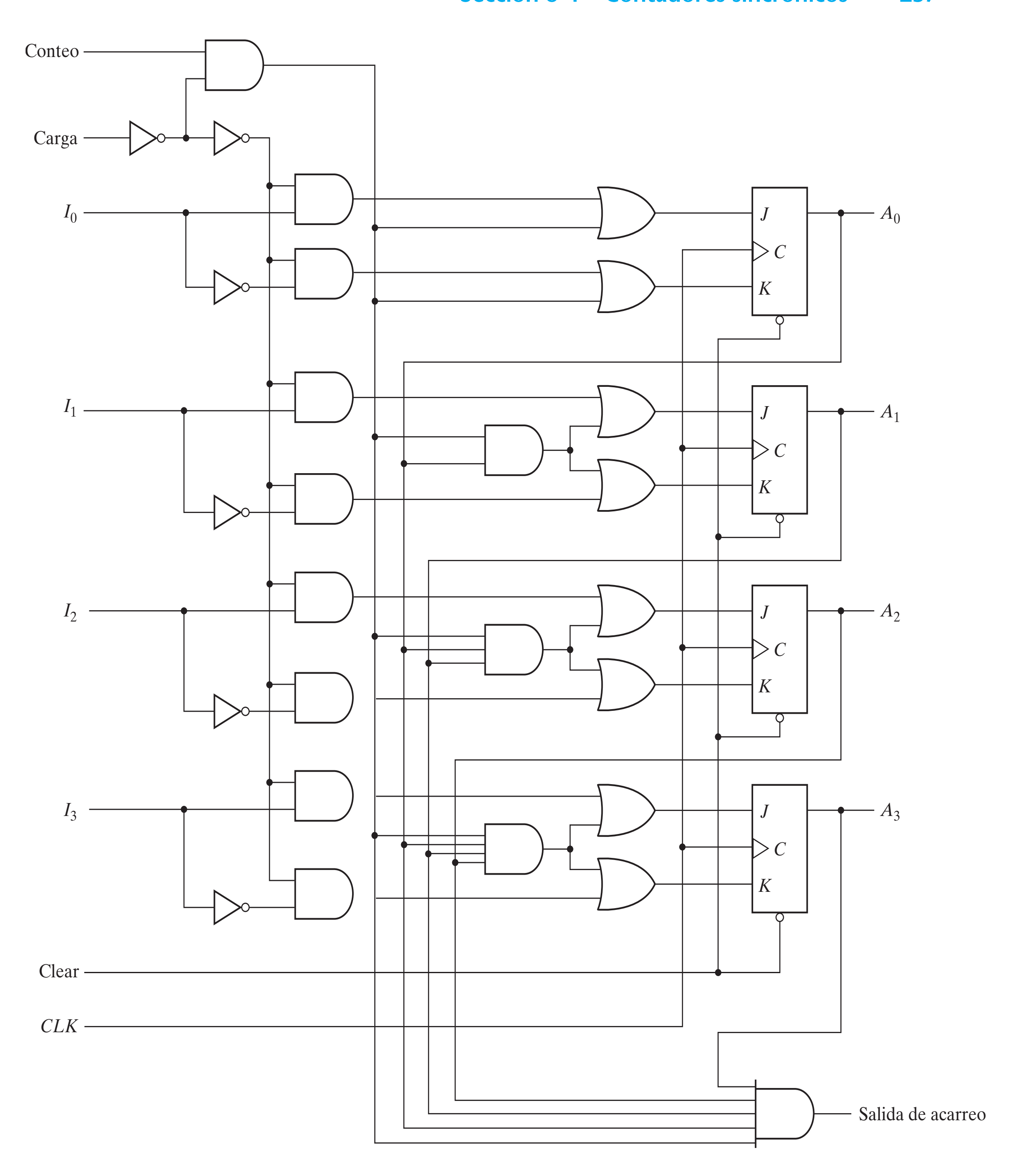


FIGURA 6-14
Contador binario de cuatro bits con carga paralela

Tabla 6-6
Tabla de función para el contador de la figura 6-14

Clear	CLK	Carga	Conteo	Función
0	X	X	X	Poner en ceros
1	↑	1	*X*	Cargar entradas
1	↑	0	1	Contar al siguiente estado binario
1	↑	0	0	Sin cambio

La salida de acarreo es 1 si todos los flip-flops son 1 y la entrada de conteo está habilitada. Ésta es la condición para complementar el flip-flop que contiene el siguiente bit significativo. La salida de acarreo es útil para expandir el contador a más de cuatro bits. La rapidez del contador aumenta si el acarreo se genera directamente a partir de las salidas de los cuatro flip-flops, pues ello reduce el retardo de generación del acarreo. Al pasar del estado 1111 a 0000, sólo hay un retardo de compuerta; en cambio, en la cadena de compuertas AND de la figura 6-12 hay cuatro retardos de compuerta. Asimismo, cada flip-flop se asocia a una compuerta AND que recibe directamente las salidas de todos los flip-flops anteriores en vez de conectar las compuertas AND en cadena.

El funcionamiento del contador se resume en la tabla 6-6. Las cuatro entradas de control: clear, CLK, carga y conteo, determinan el siguiente estado. La entrada clear es asincrónica y, si es 0, hace que el contador se despeje (se ponga en ceros) sin importar si hay pulsos de reloj u otras entradas. Esto se indica en la tabla con las entradas X, que representan condiciones de indiferencia para las demás entradas. La entrada clear debe estar en el estado 1 para que se efectúen todas las demás operaciones. Si la entrada de carga y la de conteo son 0, las salidas no cambian, aunque se apliquen pulsos de reloj. Una entrada de carga 1 causa una transferencia de las entradas I_0-I_3 al registro durante un borde positivo del reloj. Los datos de entrada se cargan en el registro sin importar qué valor tenga la entrada de conteo, porque ésta se inhibe cuando la entrada de carga está habilitada. La entrada de carga debe ser 0 para que la entrada de conteo controle el funcionamiento del contador.

Podemos usar un contador con carga paralela para generar cualquier sucesión de conteo deseada. La figura 6-15 muestra dos formas de usar un contador con carga paralela para generar un conteo BCD. En ambos casos, el control de conteo se pone en 1 para habilitar el conteo a través de la entrada CLK. Recuerde también que el control de entrada inhibe el conteo y que la operación de despeje es independiente de las demás entradas de control.

La compuerta AND de la figura 6-15a) detecta la ocurrencia del estado 1001. Inicialmente, el contador se pone en ceros y luego las entradas clear y conteo se ponen en 1 para que el contador esté siempre activo. En tanto la salida de la compuerta AND sea 0, cada borde positivo del reloj incrementará el contador en uno. Cuando la salida llegue a la cuenta de 1001, tanto A_0 como A_3 serán 1, y la salida de la compuerta AND será 1. Esta condición activa la entrada de carga; entonces, cuando llegue el siguiente borde de reloj, el registro no contará, sino que se cargará de sus cuatro entradas de datos. Puesto que esas cuatro entradas están conectadas a 0 lógico, se cargará un valor de ceros en el registro después de la cuenta de 1001. Así pues, el circuito efectúa el conteo de 0000 hasta 1001 y luego vuelve a 0000, como debe hacer un contador BCD.

En la figura 6-15b), la compuerta NAND detecta la cuenta de 1010, y en ese mismo instante el registro se despeja. La cuenta de 1010 no tiene oportunidad de durar un tiempo aprecia-

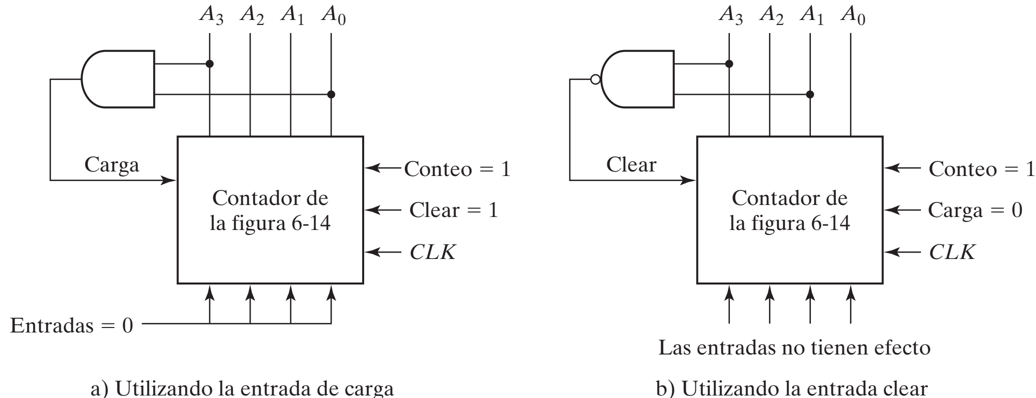


FIGURA 6-15
Dos formas de construir un contador BCD empleando un contador con carga paralela

ble, porque el registro pasa inmediatamente a 0. Hay un pico momentáneo en la salida A_0 cuando la cuenta pasa de 1010 a 1011 e inmediatamente a 0000. Este pico momentáneo podría ser indeseable, y es por ello que no se recomienda esta configuración. Si el contador tiene una entrada clear sincrónica, sería posible despejar el contador con el reloj después de que se presenta la cuenta 1001.

6-5 OTROS CONTADORES

Es posible diseñar contadores que generen cualquier sucesión de estados deseada. Un contador de división entre N (también llamado contador módulo-N) pasa por una sucesión repetida de N estados. Dicha sucesión podría seguir el conteo binario o podría ser cualquier otra sucesión arbitraria. Se emplean contadores para generar señales de temporización que controlan la sucesión de operaciones de un sistema digital. También es posible construir contadores con registros de desplazamiento. En esta sección, se presentarán unos cuantos ejemplos de contadores no binarios.

Contador con estados no utilizados

Un circuito con n flip-flops tiene 2^n estados binarios. Hay ocasiones en que un circuito secuencial utiliza menos de este máximo número posible de estados. Los estados que no se usan para especificar el circuito secuencial no se incluyen en la tabla de estados. Al simplificar las ecuaciones de entrada, los estados no utilizados podrían tratarse como condiciones de indiferencia, o asignárseles siguientes estados específicos. Una vez diseñado y construido el circuito, una interferencia externa podría hacer que el circuito quede en uno de los estados no utilizados. En tal caso, será necesario asegurarse de que el circuito pase en algún momento a uno de los estados válidos para poder reanudar su operación normal. De lo contrario, si el circuito secuencial circula entre estados no utilizados, no habrá forma de que regrese a la suce-

Tabla 6-7
Tabla de estados de un contador

Estado actual			Siguiente estado			Entradas de flip-flops					
A	*B*	*C*	*A*	*B*	*C*	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	1	0	0	1	X	X	1	0	X
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	0	0	0	X	1	X	1	0	X

sión diseñada de transiciones de estado. Si los estados no utilizados se tratan como condiciones de indiferencia, entonces, una vez que el circuito se haya diseñado, se le deberá investigar para determinar el efecto de los estados no utilizados. El estado que sigue a un estado no utilizado se determina efectuando un análisis del circuito ya diseñado.

Como ilustración, consideremos el contador especificado en la tabla 6-7. El conteo tiene una sucesión repetida de seis estados, en la que los flip-flops *B* y *C* repiten el conteo binario 00, 01, 10, y el flip-flop *A* alterna entre 0 y 1 cada tres conteos. La sucesión de conteo no es binaria directa, y dos estados, 011 y 111, no están incluidos en el conteo. La decisión de usar flip-flops *JK* da pie a las condiciones de entrada de flip-flop que se especifican en la tabla. Las entradas K_B y K_C sólo tienen unos y cruces en sus columnas, de modo que estas entradas siempre son 1. Las demás ecuaciones de entrada de flip-flop se simplifican utilizando los minitérminos 3 y 7 como condiciones de indiferencia. Las ecuaciones simplificadas son

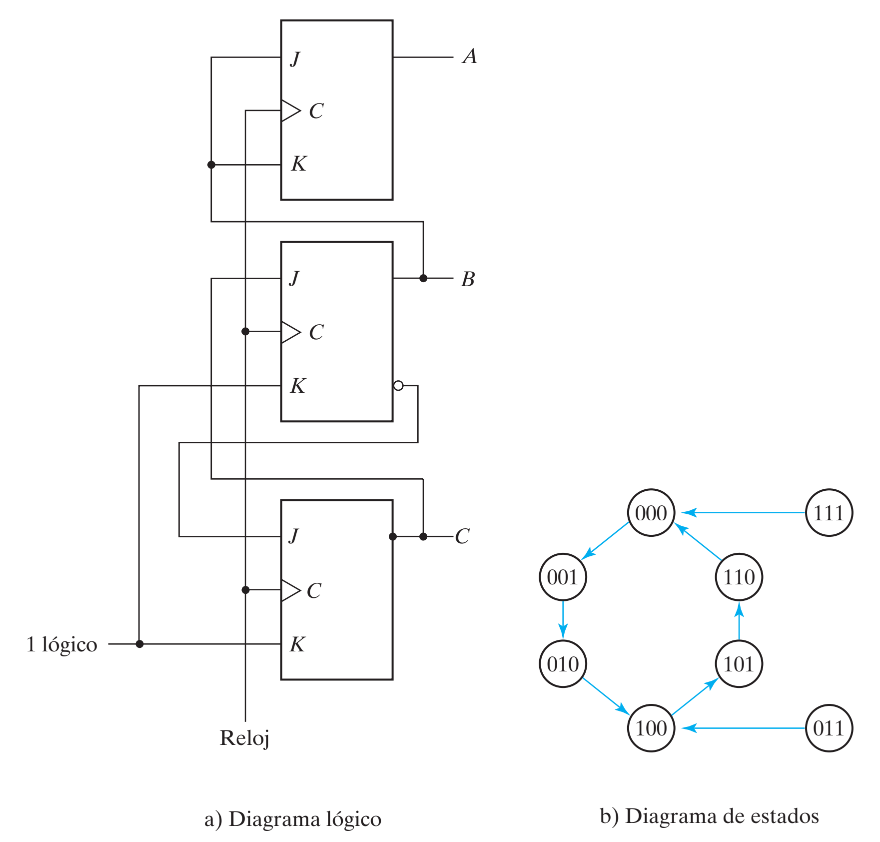
$$J_A = B \qquad K_A = B$$

$$J_B = C \qquad K_B = 1$$

$$J_C = B' \qquad K_C = 1$$

El diagrama lógico del contador se reproduce en la figura 6-16a). Puesto que hay dos estados no utilizados, analizaremos el circuito para determinar su efecto. Si el circuito llega a estar en el estado 011 debido a un error de señal, pasará al estado 100 después de la aplicación de un pulso de reloj. Esto se averigua por inspección del diagrama lógico, observando que, cuando $B = 1$, el siguiente borde de reloj complementa a *A* y pone a *C* en 0, y cuando $C = 1$, el siguiente borde de reloj complementa a *B*. De forma similar, podemos ver que el estado que sigue a 111 es 000.

En la figura 6-16b) se observa el diagrama de estados que incluye el efecto de los estados no utilizados. Si el circuito llega a quedar en uno de los estados no utilizados debido a una interferencia externa, el siguiente pulso de conteo lo transferirá a uno de los estados válidos y el circuito seguirá contando correctamente. Así pues, este circuito tiene autocorrección. Un contador tiene autocorrección si, en caso de quedar en uno de los estados no utilizados, llega a la sucesión normal de conteo después de uno o más pulsos de reloj.



a) Diagrama lógico

b) Diagrama de estados

FIGURA 6-16
Contador con estados no utilizados

Contador anular

Las señales de temporización que controlan la sucesión de operaciones de un sistema digital se pueden generar con un registro de desplazamiento o con un contador provisto de decodificador. Un *contador anular* (o de anillo) es un registro de desplazamiento circular en el que sólo un flip-flop está establecido en cualquier instante dado; los demás están despejados. El bit solitario se desplaza de un flip-flop al siguiente para producir la sucesión de señales de temporización. En la figura 6-17a) se aprecia un registro de desplazamiento de cuatro bits conectado como contador anular. El valor inicial del registro es 1000. El bit 1 se desplaza a la derecha con cada pulso de reloj y al llegar a T_3 circula de vuelta a T_0. Cada flip-flop está en el estado 1 una vez cada cuatro ciclos de reloj y produce una de las cuatro señales de temporización que se indican en la figura 6-17c). Cada salida se convierte en 1 después de la transición de borde negativo de un pulso de reloj y sigue siendo 1 durante el siguiente ciclo de reloj.

Las señales de temporización también pueden generarse con un contador de dos bits que pasa por cuatro estados distintos. El decodificador que se ilustra en la figura 6-17b) decodifica los cuatro estados del contador y genera la sucesión requerida de señales de temporización.

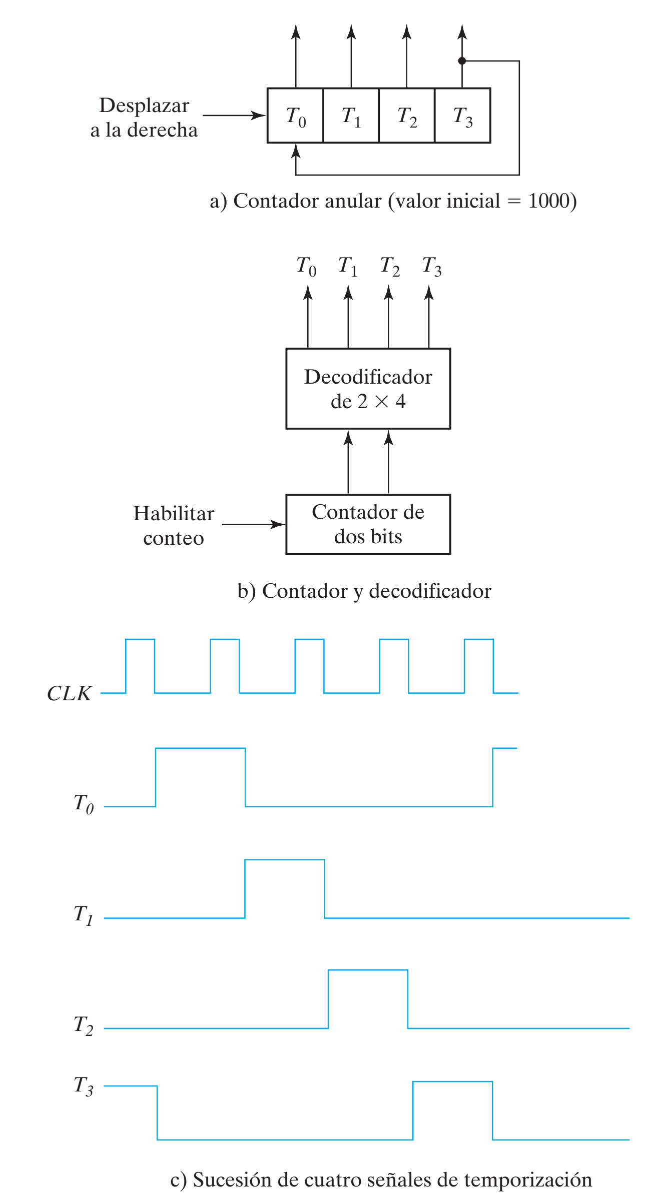


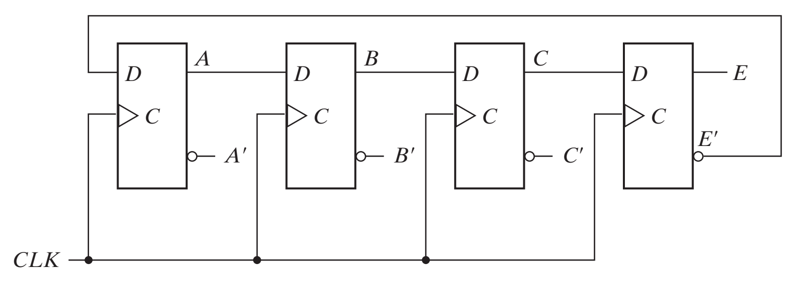
FIGURA 6-17
Generación de señales de temporización

Para generar 2^n señales de temporización, se requiere un registro de desplazamiento con 2^n flip-flops o bien un contador binario de n bits junto con un decodificador de n a 2^n líneas. Por ejemplo, podemos generar 16 señales de temporización con un registro de desplazamiento de 16 bits conectado como contador anular, o con un contador binario de cuatro bits y un decodificador de 4 a 16 líneas. En el primer caso, se necesitan 16 flip-flops. En el segundo, necesitaremos cuatro flip-flops y 16 compuertas AND de cuatro entradas para el decodificador. También

es posible generar las señales de temporización con una combinación de un registro de desplazamiento y un decodificador. En este caso, el número de flip-flops es menor que con un contador anular, y el decodificador sólo requiere compuertas de dos entradas. La combinación se denomina *contador Johnson*.

Contador Johnson

Un contador anular de *k* bits circula un solo bit entre los flip-flops para producir *k* estados distinguibles. El número de estados puede duplicarse si el registro de desplazamiento se conecta como contador anular *con extremo conmutado*. Un contador anular con extremo conmutado es un registro de desplazamiento circular en el que la salida de complemento del último flip-flop está conectada a la entrada del primer flip-flop. La figura 6-18a) muestra un registro de desplazamiento de este tipo. La conexión circular se efectúa entre la salida de complemento del flip-flop de la extrema derecha y la entrada del flip-flop de la extrema izquierda. El registro desplaza su contenido una vez a la derecha con cada pulso de reloj y, al mismo tiempo, el valor complementado del flip-flop *E* se transfiere al flip-flop *A*. Empezando en el estado despejado, el contador anular con extremo conmutado pasa por una sucesión de ocho estados, la cual se representa en la figura 6-18b). En general, un contador anular con extremo conmutado de *k* bits pasa por una sucesión de 2*k* estados. Partiendo de ceros, cada operación de desplazamiento inserta unos por la izquierda hasta que el registro queda lleno de unos. A continuación, se insertan ceros por la izquierda hasta que el registro vuelve a estar lleno de ceros.



a) Contador anular con extremo conmutado de cuatro etapas

Número sucesivo	Salidas de los flip-flops *A*	*B*	*C*	*E*	Compuerta AND requerida para la salida
1	0	0	0	0	$A'E'$
2	1	0	0	0	AB'
3	1	1	0	0	BC'
4	1	1	1	0	CE'
5	1	1	1	1	AE
6	0	1	1	1	$A'B$
7	0	0	1	1	$B'C$
8	0	0	0	1	$C'E$

b) Sucesión de conteo y decodificación requerida

FIGURA 6-18
Construcción de un contador Johnson

Un contador Johnson es un contador anular con extremo conmutado de k bits provisto de $2k$ compuertas decodificadoras para generar salidas correspondientes a $2k$ señales de temporización. Las compuertas decodificadoras no se indican en la figura 6-18, pero se especifican en la última columna de la tabla. Las ocho compuertas AND que se registran en la tabla, conectadas al circuito, completan la construcción del contador Johnson. Puesto que cada compuerta se habilita durante una sucesión dada de estados, las salidas de las compuertas generan ocho señales sucesivas de temporización.

La decodificación de un contador anular con extremo conmutado de k bits para obtener $2k$ señales de temporización sigue un patrón regular. El estado de puros ceros se decodifica tomando el complemento de las salidas de los dos flip-flops de los extremos. El estado de puros unos se decodifica tomando las salidas normales de los dos flip-flops de los extremos. Todos los demás estados se decodifican a partir de un patrón 1, 0 o 0, 1 adyacente en la sucesión. Por ejemplo, la sucesión 7 tiene un patrón 0, 1 adyacente en los flip-flops B y C. La salida decodificada se obtiene tomando el complemento de B y la salida normal de C, es decir, $B'C$.

Una desventaja del circuito de la figura 6-18a) es que si llega a estar en un estado no utilizado, persistirá en pasar de un estado no válido a otro y nunca llegará a un estado válido. Esto se corrige modificando el circuito a modo de evitar esta condición indeseable. Un procedimiento de corrección consiste en desconectar la salida del flip-flop B que va a la entrada D del flip-flop C, y habilitar la entrada del flip-flop C con la función

$$D_C = (A + C)B$$

donde D_C es la ecuación para la entrada D del flip-flop C.

Podemos construir contadores Johnson para cualquier número de sucesiones de temporización. El número de flip-flops requeridos es la mitad del número de señales de temporización. El número de compuertas decodificadoras es igual al número de señales de temporización, y sólo se necesitan compuertas de dos entradas.

6-6 HDL PARA REGISTROS Y CONTADORES

Los registros y contadores se describen en HDL en el nivel de comportamiento o en el estructural. En el primero, el registro se especifica describiendo las diversas operaciones que realiza, de forma similar a una tabla de función. Una descripción en el nivel estructural muestra el circuito en términos de una colección de componentes como compuertas, flip-flops y multiplexores. Se crean ejemplares de los diversos componentes para formar una descripción jérárquica del diseño, similar a una representación de un diagrama lógico. Utilizaremos tres circuitos de este capítulo para ilustrar los dos tipos de descripciones.

Registro de desplazamiento

El registro de desplazamiento universal que se presentó en la sección 6-2 es un registro de desplazamiento bidireccional con carga paralela. En la tabla 6-6 se especifican las cuatro operaciones con reloj que se efectúan con el registro. El registro también se puede despejar asincrónicamente. La descripción del comportamiento de un registro de desplazamiento universal de cuatro bits se ilustra en el ejemplo HDL 6-1. Hay dos entradas de selección, dos entradas en serie, una entrada en paralelo de cuatro bits y una salida en paralelo de cuatro bits. El bloque **always** describe las cinco operaciones que es posible efectuar con el registro. La en-

Ejemplo HDL 6-1

```
//Descripción del comportamiento de un
//registro de desplazamiento universal
// figura 6-7 y tabla 6-3
module shftreg (s1,s0,Pin,lfin,rtin,A,CLK,Clr);
   input s1,s0;                   //Seleccionar entradas
   input lfin, rtin;              //Entradas en serie
   input CLK,Clr;                 //Reloj y Clear
   input [3:0] Pin;               //Entrada paralela
   output [3:0] A;                //Salida del registro
   reg [3:0] A;
   always @ (posedge CLK or negedge Clr)
     if (~Clr) A = 4'b0000;
      else
        case ({s1,s0})
         2'b00: A = A;            //Sin cambio
         2'b01: A = {rtin,A[3:1]};  //Desplazamiento a la derecha
         2'b10: A = {A[2:0],lfin};  //Desplazamiento a la izquierda
         2'b11: A = Pin;          //Entrada de carga paralela
        endcase
endmodule
```

trada Clr despeja (pone en ceros) el registro asincrónicamente con una señal negativa. Clr debe estar alta para que el registro responda al borde positivo del reloj. Las cuatro operaciones con reloj del registro se determinan a partir de los valores de las dos entradas de selección en el enunciado **case** (s1 y s0 se concatenan en un vector de dos bits después de la palabra clave **case**). El desplazamiento se especifica con la concatenación de la entrada en serie y tres flip-flops. Por ejemplo, el enunciado

```
A = {rtin, A[3:1]}
```

especifica una concatenación de la entrada en serie para desplazamiento a la derecha (rtin) y los flip-flops $A3$, $A2$ y $A1$ para formar un número de cuatro bits, que se transfiere a A[3:0]. Esto produce una operación de desplazamiento a la derecha. Considere que sólo se ha descrito la función del circuito, independientemente del hardware específico.

Podemos describir la estructura del registro remitiéndonos al diagrama lógico de la figura 6-7. Ese diagrama indica que el registro se construye con cuatro multiplexores y cuatro flip-flops *D*. La descripción estructural del registro se muestra en el ejemplo HDL 6-2. El ejemplo tiene dos módulos. El primero declara las entradas y las salidas, y luego crea ejemplares para las etapas del registro. Los cuatro ejemplares creados especifican las interconexiones entre las cuatro etapas y proporcionan los pormenores de construcción del registro especificados en el diagrama lógico. El segundo módulo tiene dos bloques **always.** El primero describe al multiplexor, y el segundo, al flip-flop. Juntos definen una etapa del registro.

Ejemplo HDL 6-2

```
//Descripción estructural de registro
//universal de desplazamiento (figura 6-7)
module SHFTREG (I,select,lfin,rtin,A,CLK,Clr);
   input [3:0] I;            //Entrada paralela
   input [1:0] select;       //Seleccionar modo
   input lfin,rtin,CLK,Clr;  //Entradas en serie, reloj, clear
   output [3:0] A;           //Salida paralela
 //Crear ejemplares para las cuatro etapas
   stage ST0 (A[0],A[1],lfin,I[0],A[0],select,CLK,Clr);
   stage ST1 (A[1],A[2],A[0],I[1],A[1],select,CLK,Clr);
   stage ST2 (A[2],A[3],A[1],I[2],A[2],select,CLK,Clr);
   stage ST3 (A[3],rtin,A[2],I[3],A[3],select,CLK,Clr);
endmodule

//Una etapa del registro de desplazamiento
module stage(i0,i1,i2,i3,Q,select,CLK,Clr);
   input i0,i1,i2,i3,CLK,Clr;
   input [1:0] select;
   output Q;
   reg Q;
   reg D;
//Multiplexor 4x1
   always @ (i0 or i1 or i2 or i3 or select)
         case (select)
            2'b00: D = i0;
            2'b01: D = i1;
            2'b10: D = i2;
            2'b11: D = i3;
         endcase
//Flip-flop D
   always @ (posedge CLK or negedge Clr)
         if (~Clr) Q = 1'b0;
         else Q = D;
endmodule
```

Contador sincrónico

El ejemplo HDL 6-3 describe el contador sincrónico con carga paralela de la figura 6-14. *Count*, *Load*, *CLK* y *Clr* (conteo, carga, reloj y despeje) son entradas que determinan el funcionamiento del registro según la función especificada en la tabla 6-6. El contador tiene cuatro entradas de datos, cuatro salidas de datos y una salida de acarreo. Esta última, *CO* se genera con un circuito combinacional y se especifica con un enunciado **assign.** $CO = 1$ cuando la cuenta llega a 15 y el contador está en el estado de conteo. Así pues, $CO = 1$ si

Ejemplo HDL 6-3

```
//Contador binario con carga paralela
//Véase la figura 6-14 y la tabla 6-6
module counter (Count,Load,IN,CLK,Clr,A,CO);
  input Count,Load,CLK,Clr;
  input [3:0] IN;              //Entrada de datos
  output CO;                   //Acarreo de salida
  output [3:0] A;              //Salida de datos
  reg [3:0] A;
  assign CO = Count & ~Load & (A == 4'b1111);
  always @ (posedge CLK or negedge Clr)
    if (~Clr) A = 4'b0000;
    else if (Load)  A = IN;
    else if (Count) A = A + 1'b1;
    else A = A;
endmodule
```

Count $= 1$, *Load* $= 0$ y $A = 1111$; en los demás casos, $CO = 0$. El bloque **always** especifica la operación a efectuar en el registro, dependiendo de los valores de *Clr, Load* y *Count*. Una señal negativa en *Clr* pone *A* en 0. Si *Clr* $= 1$, una de tres operaciones se ejecutan durante un borde positivo del reloj. Los enunciados **if, else if** y **else** toman las decisiones como sigue:

```
    if Clr = 0                                   Poner A en ceros
else if (Clr = 1 and) Load = 1                   Cargar entradas en A
else if (Clr = 1 and Load = 0 and) Count = 1     Incrementar A
   else (Clr = 1 and Load = 0 and Count = 0)     Sin cambio en A
```

La jerarquía implícita en los enunciados **if-else** se ajusta a la precedencia especificada en la tabla 6-6.

Contador de rizo

En el ejemplo HDL 6-4 se muestra la descripción estructural de un contador de rizo. El primer módulo crea ejemplares de cuatro flip-flops complementadores que se definen en el segundo módulo como `CF(Q,CLK,Reset)`. El reloj (entrada C) del primer flip-flop se conecta a la entrada externa *Count* (*Count* sustituye a *CLK* en *F*0). La entrada de reloj del segundo flip-flop está conectada a la salida del primero (*A*0 sustituye a *CLK* en *F*1). De forma similar, el reloj de cada uno de los otros flip-flops está conectado a la salida del flip-flop anterior. Así, los flip-flops se encadenan para formar un contador de rizo, como se advierte en la figura 6-8b).

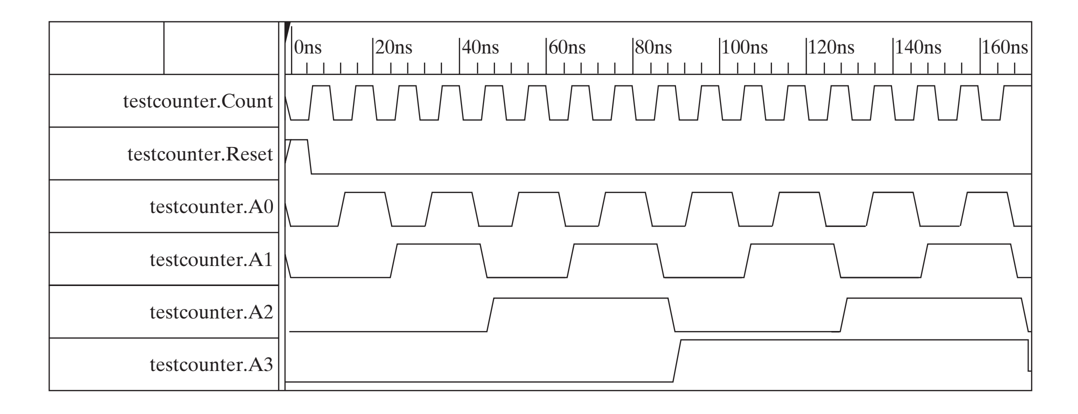
El segundo módulo describe un flip-flop complementador con retardo. El circuito se construye conectando la salida de complemento a la entrada *D*. Se incluye una entrada de restable-

Ejemplo HDL 6-4

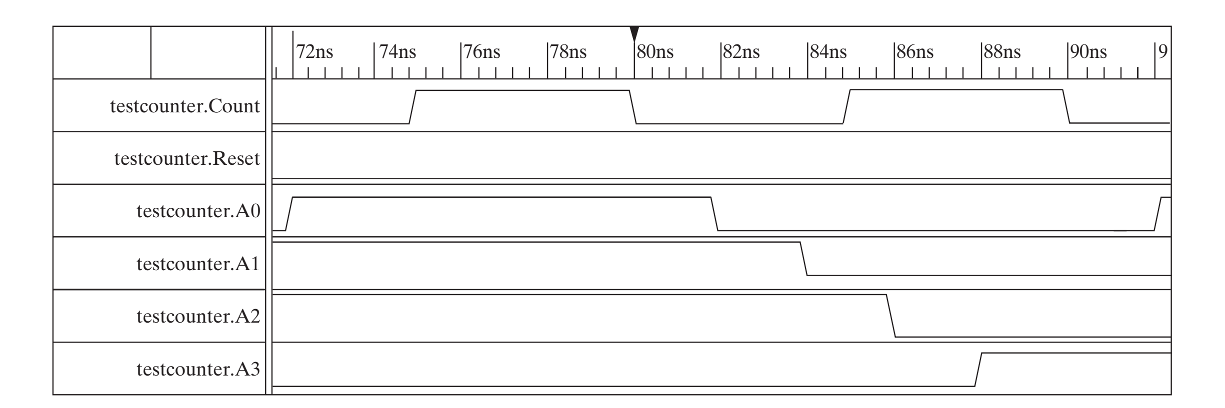
```
//Contador de rizo (Véase la figura 6-8b))
module ripplecounter (A0,A1,A2,A3,Count,Reset);
   output A0,A1,A2,A3;
   input Count,Reset;
//Crear ejemplar de flip-flop complementador
   CF F0 (A0,Count,Reset);
   CF F1 (A1,A0,Reset);
   CF F2 (A2,A1,Reset);
   CF F3 (A3,A2,Reset);
endmodule

//Flip-flop complementador con retardo
//Entrada al flip-flop D = Q'
module CF (Q,CLK,Reset);
   output Q;
   input CLK,Reset;
   reg Q;
   always @ (negedge CLK or posedge Reset)
      if (Reset) Q = 1'b0;
      else  Q = #2 (~Q);     // Retardo de 2 unidades de tiempo
endmodule

//Estímulo para probar el contador de rizo
module testcounter;
   reg Count;
   reg Reset;
   wire A0,A1,A2,A3;
//Crear ejemplar de contador de rizo
   ripplecounter RC (A0,A1,A2,A3,Count,Reset);
always
    #5 Count = ~Count;
initial
 begin
      Count = 1'b0;
      Reset = 1'b1;
   #4 Reset = 1'b0;
   #165 $finish;
 end
endmodule
```

cimiento en el flip-flop para poder iniciar el contador. Los simuladores HDL no proporcionan valores de salida si no se les asigna un valor inicial. Se asigna al flip-flop un retardo de dos unidades de tiempo, desde el momento en que se aplica el reloj hasta el momento en que el flip-flop se complementa. Esto se especifica con el enunciado $Q = \#2\ (\sim Q)$.

El tercer módulo del ejemplo 6-4 genera un estímulo para simular y probar el contador de rizo. El enunciado **always** genera un reloj con un ciclo de 10 unidades de tiempo. Los flip-flops se disparan con el borde negativo del reloj, que se da en $t = 10, 20, 30$, etcétera. Las formas de onda que se obtienen de esta simulación se distinguen en la figura 6-19. *Count* se vuelve negativa cada 10 ns. *A*0 se complementa con cada borde negativo de *Count* pero con un retardo de 2 ns. Cada flip-flop se complementa cuando el flip-flop precedente cambia de 1 a 0. Después de $t = 80$ ns, los cuatro flip-flops se complementan porque el contador cambia de 0111 a 1000. Cada salida se retarda 2 ns y, a causa de ello, *A*3 cambia de 0 a 1 en $t = 88$ ns, y de 1 a 0 a los 168 ns.



a) De 0 a 170 ns



b) De 70 a 92 ns

FIGURA 6-19
Salida de simulación del ejemplo HDL 6-4

PROBLEMAS

6-1 Incluya una compuerta NAND de dos entradas con el registro de la figura 6-1 y conecte la salida de la compuerta a las entradas *C* de todos los flip-flops. Una entrada de la compuerta NAND recibe los pulsos de reloj del generador de reloj, y la otra entrada de la compuerta se encarga de controlar la carga en paralelo. Explique el funcionamiento del registro modificado.

6-2 Incluya una entrada de despeje sincrónica para el registro de la figura 6-2. El registro modificado tendrá una capacidad de carga en paralelo y una capacidad de despeje sincrónico. El registro se despeja (pone en ceros) sincrónicamente cuando el reloj tiene una transición positiva y la entrada de despeje es 1.

6-3 ¿Qué diferencia hay entre transferencia en serie y en paralelo? Explique cómo convertir datos en serie a paralelo y datos en paralelo a datos en serie. ¿Qué tipo de registro se necesita?

6-4 El contenido de un registro de cuatro bits es inicialmente 1101. El registro se desplaza seis veces a la derecha, siendo la entrada en serie 101101. ¿Qué contiene el registro después de cada desplazamiento?

6-5 El registro universal de desplazamiento de cuatro bits mostrado en la figura 6-7 se encierra en un paquete de CI.

a) Dibuje un diagrama de bloques del circuito integrado que señale todas las entradas y salidas. Incluya dos entradas para la alimentación eléctrica.

b) Dibuje un diagrama de bloques empleando dos CI para producir un registro de desplazamiento universal de ocho bits.

6-6 Diseñe un registro de desplazamiento de cuatro bits con carga paralela empleando flip-flops *D*. Hay dos entradas de control: desplazar y cargar. Cuando desplazar $= 1$, el contenido del registro se desplaza una posición. Se transfieren nuevos datos al registro cuando cargar $= 1$ y desplazar $= 0$. Si ambas entradas de control son 0, el contenido del registro no cambia.

6-7 Dibuje el diagrama lógico de un registro de cuatro bits con cuatro flip-flops *D* y cuatro multiplexores 4×1, con entradas de selección de modo s_1 y s_0. El registro opera según la siguiente tabla de función:

s_1	s_0	Operación del registro
0	0	Sin cambio
0	1	Complementar las cuatro salidas
1	0	Poner el registro en ceros (sincrónico con el reloj)
1	1	Cargar datos en paralelo

6-8 El sumador en serie de la figura 6-6 usa dos registros de cuatro bits. El registro *A* contiene el número binario 0101, y el registro *B*, 0111. El flip-flop de acarreo se restablece inicialmente en 0. Numere los valores binarios que están en el registro *A* y en el flip-flop de acarreo después de cada desplazamiento.

6-9 En la sección 6-2 se describieron dos formas de implementar un sumador en serie ($A + B$). Es necesario modificar los circuitos para convertirlos en restadores en serie ($A - B$).

a) Utilizando el circuito de la figura 6-5, indique los cambios necesarios para obtener $A +$ complemento a dos de *B*.

b) Utilizando el circuito de la figura 6-6, indique los cambios requeridos modificando la tabla 6-2, de un circuito sumador a uno restador. (Véase el problema 4-12.)

6-10 Diseñe un complementador a dos en serie con un registro de desplazamiento y un flip-flop. El número binario se desplaza hacia afuera por un lado y su complemento a dos se desplaza hacia adentro por el otro lado del registro de desplazamiento.

6-11 Un contador binario de rizo usa flip-flops que se disparan con el borde positivo del reloj. ¿Cuál será el conteo si a) las salidas normales de los flip-flops se conectan al reloj y b) las salidas de complemento de los flip-flops se conectan al reloj?

6-12 Dibuje el diagrama lógico de un sumador binario de rizo de cuatro bits de cuenta regresiva utilizando a) flip-flops que se disparan con el borde positivo del reloj y b) flip-flops que se disparan con el borde negativo del reloj.

6-13 Demuestre que es posible construir un contador BCD de rizo empleando un contador binario de rizo de cuatro bits con despeje asincrónico y una compuerta NAND que detecta la ocurrencia de la cuenta 1010.

6-14 ¿Cuántos flip-flops se complementarán en un contador binario de rizo de 10 bits para llegar a la siguiente cuenta después de la cuenta?: a) 1001100111; b) 0011111111; c) 1111111111.

6-15 Un flip-flop tiene un retardo de 5 ns desde el momento en que se da el borde de reloj hasta el momento en que la salida se complementa. ¿Qué retardo máximo tendría un contador binario de rizo de 10 bits que usara esos flip-flops? ¿Con qué frecuencia máxima puede operar el contador de manera confiable?

6-16 El contador BCD de rizo que se representa en la figura 6-10 tiene cuatro flip-flops y 16 estados, de los cuales sólo se usan 10. Analice el circuito y determine el siguiente estado para cada uno de los otros seis estados no utilizados. ¿Qué sucederá si una señal de ruido hace que el circuito pase a uno de los estados no utilizados?

6-17 Diseñe un contador binario sincrónico de cuatro bits con flip-flops *D*.

6-18 ¿Qué operación se efectúa en el contador ascendente-descendente de la figura 6-13 cuando ambas entradas, arriba y abajo, están habilitadas? Modifique el circuito de modo que cuando ambas entradas sean 1, el contador no cambie de estado, sino que permanezca en la misma cuenta.

6-19 Las ecuaciones de entrada de flip-flops para un contador BCD construido con flip-flops *T* se incluyen en la sección 6-4. Obtenga las ecuaciones de entrada para un contador BCD construido con a) flip-flops *JK* y b) flip-flops *D*. Compare los tres diseños para determinar cuál es el más eficiente.

6-20 Encierre el contador binario con carga paralela de la figura 6-14 en un diagrama de bloques que muestre todas las entradas y salidas.

a) Muestre las conexiones de cuatro de esos bloques para formar un contador de 16 bits con carga paralela.

b) Construya un contador binario que cuente desde 0 hasta 64 binario.

6-21 El contador de la figura 6-14 tiene dos entradas de control —Cargar (*L*) y Conteo c)— y una entrada de datos (I_i).

a) Deduzca las ecuaciones de entrada de flip-flops para *J* y *K* de la primera etapa, en términos de *L*, *C* e *I*.

b) En la figura P6-21 se observa el diagrama lógico de la primera etapa de un circuito integrado equivalente (74161). Compruebe que este circuito sea equivalente al de a).

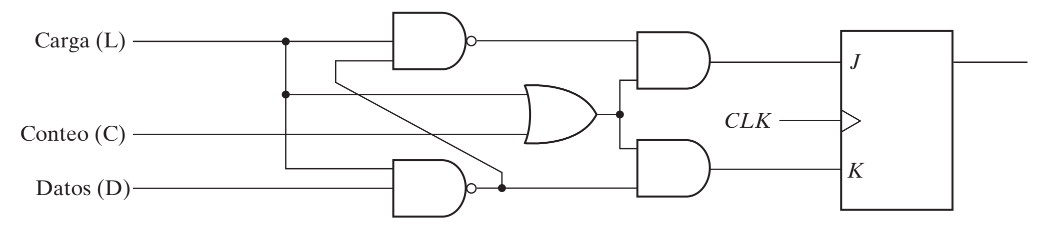


FIGURA P6-21

6-22 Utilizando el circuito de la figura 6-14, dé tres alternativas para un contador mod-12:

a) Utilizando una compuerta AND y la entrada de carga.

b) Utilizando el acarreo de salida.

c) Utilizando una compuerta NAND y la entrada de despeje asincrónico.

6-23 Diseñe un circuito de temporización que genere una señal de salida que se mantenga encendida durante exactamente ocho ciclos de reloj. Una señal de inicio hace que la salida pase al estado 1; después de ocho ciclos de reloj, la señal vuelve al estado 0.

6-24 Diseñe con flip-flops T un contador que pase por la siguiente sucesión binaria repetida: 0, 1, 3, 7, 6, 4. Demuestre que si los estados binarios 010 y 101 se consideran condiciones de indiferencia, el contador podría no funcionar correctamente. Encuentre una forma de corregir el diseño.

6-25 Es necesario generar seis señales repetidas de temporización T_0 a T_5 similares a las que se indican en la figura 6-17c). Diseñe el circuito utilizando:

a) Únicamente flip-flops.

b) Un contador y un decodificador.

6-26 Un sistema digital tiene un generador de reloj que produce pulsos con una frecuencia de 80 MHz. Diseñe un circuito que genere un reloj con un tiempo de ciclo de 50 ns.

6-27 Diseñe un contador que siga esta sucesión binaria repetida: 0, 1, 2, 3, 4, 5, 6. Use flip-flops JK.

6-28 Diseñe un contador que siga esta sucesión binaria repetida: 0, 1, 2, 4, 6. Use flip-flops D.

6-29 Numere los ocho estados no utilizados del contador de anillo con extremo conmutado de la figura 6-18a).

Determine el siguiente estado para cada uno de estos estados y demuestre que, si el contador llega a estar en un estado no válido, no volverá a un estado válido. Modifique el circuito como se recomienda en el texto y demuestre que el contador produce la misma sucesión de estados y que el circuito llega a un estado válido desde cualquiera de los estados no utilizados.

6-30 Demuestre que un contador Johnson con n flip-flops produce una sucesión de $2n$ estados. Numere los 10 estados producidos con cinco flip-flops y los términos booleanos de cada una de las diez salidas de compuerta AND.

6-31 Escriba las descripciones HDL de comportamiento y estructural del registro de cuatro bits de la figura 6-1.

6-32 a) Escriba la descripción HDL del comportamiento de un registro de cuatro bits con carga paralela y despeje asincrónico.

b) Escriba la descripción HDL estructural del registro de cuatro bits con carga paralela de la figura 6-2. Utilice un multiplexor 2×1 para las entradas de flip-flops. Incluya una entrada de despeje asincrónico.

c) Verifique ambas descripciones con un conjunto de pruebas.

6-33 Se usa el programa de estímulo siguiente para simular el contador binario con carga paralela descrito en el ejemplo HDL 6-3. Examine el programa y prediga qué salida tendrá el contador y el acarreo entre $t = 0$ y $t = 155$ ns.

```
//Estímulo para probar el contador
//del ejemplo 6-3
     module testcounter;
       reg Count, Load, CLK, Clr;
       reg [3:0] IN;
       wire C0;
       wire [3:0] A;
       counter cnt (Count, Load, IN, CLK, Clr, A, CO);
       always
           #5 CLK = ~CLK;
         initial
          begin
           Clr = 0;
           CLK = 1;
           Load = 0; Count = 1;
             #5  Clr = 1;
             #50 Load = 1; IN = 4'b1100;
             #10 Load = 0;
             #70 Count = 0;
             #20 $finish;
          end
     endmodule
```

6-34 Escriba la descripción HDL del comportamiento de un registro de desplazamiento de cuatro bits (figura 6-3).

6-35 Escriba las descripciones HDL de comportamiento y estructural del contador arriba-abajo de cuatro bits cuyo diagrama lógico aparece en la figura 6-13.

6-36 Escriba la descripción HDL del comportamiento de un contador arriba-abajo de cuatro bits con carga paralela utilizando las siguientes entradas de control:

a) El contador tiene tres entradas de control para las tres operaciones: Arriba, Abajo y Cargar. El orden de precedencia es: Cargar, Arriba y Abajo.

b) El contador tiene dos entradas de selección para especificar cuatro operaciones: Arriba, Abajo, Cargar y sin cambio.

6-37 Escriba la descripción HDL de un contador anular de ocho bits similar al de la figura 6-17a).

6-38 Escriba la descripción HDL de un contador anular con extremo conmutado de cuatro bits (figura 6-18a).

6-39 Escriba las descripciones HDL de comportamiento y estructural del contador de la figura 6-16.

REFERENCIAS

1. MANO, M. M. y C. R. KIME. 2000. *Logic and Computer Design Fundamentals*. 2a. ed. Upper Saddle River, NJ: Prentice-Hall.
2. NELSON V. P., H. T. NAGLE, J. D. IRWIN y B. D. CARROLL. 1995. *Digital Logic Circuit Analysis and Design*. Upper Saddle River, NJ: Prentice-Hall.
3. HAYES, J. P. 1993. *Introduction to Digital Logic Design*. Reading, MA: Addison-Wesley.
4. WAKERLY, J. F. 2000. *Digital Design: Principles and Practices*. 3a. ed. Upper Saddle River, NJ: Prentice-Hall.
5. DIETMEYER, D. L. 1988. *Logic Design of Digital Systems*. 3a. ed. Boston: Allyn Bacon.
6. GAJSKI, D. D. 1997. *Principles of Digital Design*. Upper Saddle River, NJ: Prentice-Hall.
7. ROTH, C. H. 1992. *Fundamentals of Logic Design*, 4a. ed. St. Paul: West.
8. KATZ, R. H. 1994. *Contemporary Logic Design*. Upper Saddle River, NJ: Prentice-Hall.
9. CILETTI, M. D. 1999. *Modeling, Synthesis and Rapid Prototyping with Verilog HDL*. Upper Saddle River, NJ: Prentice-Hall.
10. BHASKER, J. 1997. *A Verilog HDL Primer*. Allentown, PA: Star Galaxy Press.
11. THOMAS, D. E. y P. R. MOORBY. 1998. *The Verilog Hardware Description Language*. 4a. ed. Boston: Kluwer Academic Publishers.
12. BHASKER, J. 1998. *Verilog HDL Synthesis*. Allentown, PA: Star Galaxy Press.
13. PALNITKAR, S. 1996. *Verilog HDL: A Guide to Digital Design and Synthesis*. SunSoft Press (Un título Prentice-Hall).

7 Memoria y lógica programable

7-1 INTRODUCCIÓN

Una unidad de memoria es un dispositivo al que se transfiere información binaria que desea almacenarse, y del que se puede obtener información que es necesario procesar. Cuando se efectúa procesamiento de datos, la información de la memoria se transfiere a registros selectos de la unidad de procesamiento. Los resultados intermedios y finales obtenidos en la unidad de procesamiento se transfieren de vuelta a la memoria para guardarse. La información binaria recibida de un dispositivo de entrada se almacena en la memoria, y la información transferida a un dispositivo de salida se toma de la memoria. Una unidad de memoria es una colección de celdas que permite almacenar una gran cantidad de información binaria.

Hay dos tipos de memorias que se usan en los sistemas digitales: *memoria de acceso aleatorio* (RAM, *random-access memory*) y *memoria de sólo lectura* (ROM, *read-only memory*). La primera acepta nueva información que se guardará para poder usarla posteriormente. El proceso de guardar información nueva en la memoria es una operación de *escritura* en memoria. El proceso de transferir desde la memoria la información en ella almacenada es una operación de *lectura* de memoria. La memoria de acceso aleatorio puede efectuar ambas operaciones, lectura y escritura. La memoria de sólo lectura únicamente puede efectuar la operación de lectura. Esto implica que ya está almacenada en ella una información binaria apropiada, que es posible recuperar o leerse en cualquier momento. Sin embargo, la información existente no puede alterarse mediante escritura porque únicamente se puede leer de la memoria de sólo lectura; no es posible escribir en ella.

La memoria de sólo lectura es un *dispositivo lógico programable*. La información binaria que se almacena en un dispositivo lógico programable se especifica de alguna manera y luego se incorpora al hardware. Llamamos a este proceso *programar* el dispositivo. La palabra "programación" en este caso se refiere a un procedimiento de hardware que especifica los bits que se insertan en la configuración de hardware del dispositivo.

La memoria de sólo lectura (ROM) es un ejemplo de dispositivo lógico programable (PLD, *programmable logic device*). Otros son el arreglo de lógica programable (PLA, *programmable logic array*), el arreglo lógico programable (PAL, *programmable array logic*) y el arreglo de

FIGURA 7-1
Diagramas convencional y de arreglo lógico para la compuerta OR

compuertas programable en el campo (FPGA, *field-programmable gate array*. Un dispositivo lógico programable es un circuito integrado con compuertas lógicas internas que se conectan mediante trayectorias electrónicas que se comportan como una especie de fusibles. En el estado original del dispositivo, todos los fusibles están intactos. Programar el dispositivo requiere "quemar" los fusibles que están en las trayectorias que es preciso eliminar para obtener la configuración de la función lógica deseada. En este capítulo presentaremos la configuración de los dispositivos lógicos programables y mencionaremos cómo se usan en el diseño de sistemas digitales.

Un dispositivo lógico programable típico podría tener desde cientos hasta millones de compuertas interconectadas por cientos o miles de trayectorias internas. Para mostrar el diagrama lógico interno de manera concisa, es necesario utilizar una simbología especial de compuertas aplicable a la lógica de arreglos. La figura 7-1 muestra los símbolos convencional y de arreglo para una compuerta OR de múltiples entradas. En lugar de dibujar varias líneas que entran en la compuerta, se dibuja una sola línea hacia la compuerta. Las líneas de entrada se dibujan perpendiculares a esa única línea y se conectan a la compuerta a través de fusibles internos. La lógica de arreglo para una compuerta AND es similar. Este tipo de representación gráfica para las entradas de compuertas se usará en todo este capítulo al dibujar diagramas de arreglos lógicos.

7-2 MEMORIA DE ACCESO ALEATORIO

Una unidad de memoria es un conjunto de celdas de almacenamiento junto con los circuitos asociados que se requieren para transferir información al y del dispositivo. El tiempo que toma transferir información a o de cualquier posición al azar deseada siempre es el mismo, de ahí el nombre *memoria de acceso aleatorio* o RAM.

Una unidad de memoria almacena información binaria en grupos de bits llamados *palabras*. Una palabra de memoria es una entidad de bits que siempre se guardan o sacan juntos, como una unidad. Una palabra de memoria es un grupo de unos y ceros y podría representar un número, una instrucción, uno o más caracteres alfanuméricos o cualquier otra información codificada en binario. Un grupo de ocho bits es un *byte*. Casi todas las memorias de computadora manejan palabras cuya longitud es un múltiplo de ocho bits. Así, una palabra de 16 bits contiene dos bytes, y una de 32 bits consta de cuatro bytes. La capacidad de una unidad de memoria por lo regular se da como el número total de bytes que es capaz de guardar.

La comunicación entre la memoria y su entorno se efectúa a través de líneas de entrada y salida de datos, líneas de selección de direcciones y líneas de control que especifican la dirección de la transferencia. En la figura 7-2 se presenta un diagrama de bloques de la unidad de memoria. Las n líneas de entrada de datos alimentan la información que se guardará en la memoria, y las n líneas de salida de datos proporcionan la información que viene de la memoria. Las k líneas de dirección especifican la palabra específica escogida, de entre muchas disponibles. Las dos entradas de control especifican la dirección de la transferencia deseada: la entrada de escritura hace que se transfieran datos binarios a la memoria; la de lectura hace que se saquen datos binarios de la memoria.

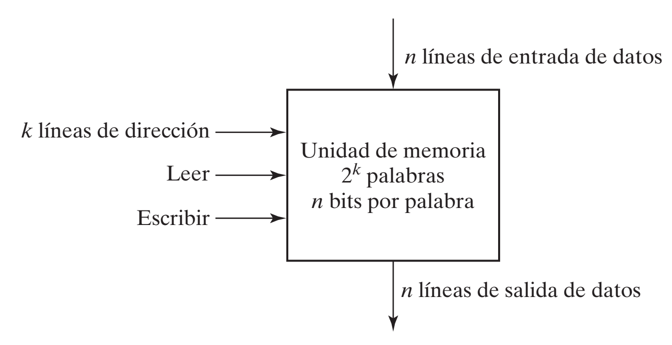


FIGURA 7-2
Diagrama de bloqes de una unidad de memoria

La unidad de memoria se especifica con el número de palabras que contiene y el número de bits que hay en cada palabra. Las líneas de dirección seleccionan una palabra específica. A cada palabra de la memoria se asigna un número de identificación, llamado *dirección*, entre 0 y $2^k - 1$, donde k es el número de líneas de dirección. La selección de una palabra específica de la memoria se efectúa aplicando los k bits de dirección a las líneas de dirección. Un decodificador acepta esta dirección y abre las trayectorias necesarias para seleccionar la palabra especificada. Las memorias varían considerablemente en cuanto a tamaño; las hay desde 1024 palabras, que requieren una dirección de 10 bits, hasta 2^{32} palabras, que requieren 32 bits de dirección. Se acostumbra especificar el número de palabras (o bytes) de la memoria con una de las letras K (kilo), M (mega) o G (giga). K es igual a 2^{10}, M es igual a 2^{20} y G es igual a 2^{30}. Así pues, $64K = 2^{16}$, $2M = 2^{21}$ y $4G = 2^{32}$.

Considere, por ejemplo, la unidad de memoria con capacidad de 1K palabras de 16 bits cada una. Puesto que $1K = 1024 = 2^{10}$ y 16 bits constituyen dos bytes, se afirma que la memoria puede dar cabida a $2048 = 2K$ bytes. La figura 7-3 muestra el posible contenido de las

Dirección de memoria		
Binaria	Decimal	Contenido de la memoria
0000000000	0	1011010101011101
0000000001	1	1010101110001001
0000000010	2	0000110101000110
	⋮	⋮
1111111101	1021	1001110100010100
1111111110	1022	0000110100011110
1111111111	1023	1101111000100101

FIGURA 7-3
Contenido de una memoria de 1024 × 16

primeras tres y las últimas tres palabras de esta memoria. Cada palabra contiene 16 bits que se dividen en dos bytes. Las palabras se reconocen por su dirección decimal, de 0 a 1023. La dirección binaria equivalente consta de 10 bits. La primera dirección se especifica con 10 ceros, y la última, con 10 unos. Ello se debe a que 1023 en binario es 1111111111. Seleccionamos una palabra de memoria por su dirección binaria. Cuando se lee o escribe una palabra, la memoria opera sobre los 16 bits como una sola unidad.

La memoria de $1K \times 16$ de la figura 7-3 tiene 10 bits en la dirección y 16 bits en cada palabra. En otro ejemplo, una memoria de $64K \times 10$ tendría 16 bits en la dirección (porque $64K = 2^{16}$) y cada palabra consistiría en 10 bits. El número de bits de dirección que una memoria necesita depende del número total de palabras que es posible guardar en la memoria, y es independiente del número de bits que hay en cada palabra. El número de bits de la dirección se determina a partir de la relación $2^k \geq m$, donde m es el número total de palabras y k es el número de bits de dirección necesarios para satisfacer la relación.

Operaciones de lectura y escritura

Las dos operaciones que efectúa una memoria de acceso aleatorio son escritura y lectura. La señal de escritura especifica una operación de transferencia hacia adentro, y la de lectura, una de transferencia hacia afuera. Al aceptar una de estas señales de control, los circuitos internos de la memoria efectúan la operación deseada.

Los pasos que deben seguirse para transferir una nueva palabra a la memoria son:

1. Aplique la dirección binaria de la localidad deseada a las líneas de dirección.
2. Aplique a las líneas de entrada de datos los bits de datos que se guardarán en la memoria.
3. Active la entrada *escribir*.

La unidad de memoria tomará entonces los bits de las líneas de datos de entrada y los almacenará en la localidad especificada por las líneas de dirección.

Los pasos que deben seguirse para sacar de la memoria una palabra almacenada son:

1. Aplique a las líneas de dirección la dirección binaria de la localidad deseada.
2. Active la entrada *leer*.

La unidad de memoria tomará entonces los bits de la localidad seleccionada por la dirección y los aplicará a las líneas de datos de salida. El contenido de la localidad seleccionada no cambia después de la lectura.

Algunos componentes de memoria que se venden comercialmente en chips de circuitos integrados ofrecen las dos entradas de control para leer y escribir en una configuración un tanto diferente. En vez de tener dos entradas individuales de leer y escribir para controlar las dos operaciones, casi todos los circuitos integrados ofrecen otras dos entradas de control: una selecciona la unidad y la otra determina la operación. Las operaciones de memoria resultado de estas entradas de control se especifican en la tabla 7-1.

La entrada de habilitar memoria (o de seleccionar chip, como también se le conoce) sirve para habilitar un chip de memoria en una implementación multichips de una memoria grande. Si habilitar memoria está inactiva, el chip no está seleccionado y no se efectúa ninguna operación. Si esa entrada está activa, la entrada leer/escribir determina la operación a efectuar.

Tabla 7-1
Entradas de control de un chip de memoria

Habilitar memoria	Leer/escribir	Operación de memoria
0	X	Ninguna
1	0	Escribir en la localidad seleccionada
1	1	Leer de la localidad seleccionada

Descripción de memoria en HDL

La memoria se modela en Verilog HDL con un arreglo de registros. Se declara con la palabra clave **reg** empleando un arreglo bidimensional. El primer índice del arreglo especifica el número de bits que tiene una palabra, y el segundo, el número de palabras que contiene la memoria. Por ejemplo, una memoria de 1024 palabras de 16 bits cada una se declara como

```
reg[15:0] memword[0:1023];
```

Esto describe un arreglo bidimensional de 1024 registros, cada uno de los cuales contiene 16 bits. El índice de memword abarca el número total de palabras que hay en la memoria y equivale a la dirección de la palabra en la memoria. Por ejemplo, `memword[512]` se refiere a la palabra de memoria de 16 bits que está en la dirección 512.

En el ejemplo HDL 7-1 se ilustra el funcionamiento de una unidad de memoria. Esta memoria tiene 64 palabras de cuatro bits cada una. Hay dos entradas de control: Enable (habilitar) y ReadWrite (leer/escribir). Las líneas DataIn (entrada de datos) y DataOut (salida de datos) tienen cuatro bits cada una. La entrada Address (dirección) debe tener seis bits (porque $2^6 = 64$). La memoria se declara como un arreglo bidimensional de registros, y Mem especi-

Ejemplo HDL 7-1

```
//Operaciones de lectura y escritura de memoria.
//El tamaño de la memoria es 64 palabras de 4 bits c/u.
 module memory (Enable,ReadWrite,Address,DataIn,DataOut);
    input  Enable,ReadWrite;
    input [3:0] DataIn;
    input [5:0] Address;
    output [3:0] DataOut;
    reg [3:0] DataOut;
    reg [3:0] Mem [0:63];          //Memoria de 64 x 4
    always @ (Enable or ReadWrite)
      if (Enable)
      if (ReadWrite)
      DataOut = Mem[Address];  //Leer
      else
      Mem[Address] = DataIn;   //Escribir
      else DataOut = 4'bz;        //Estado de alta impedancia
endmodule
```

fica la dirección de las 64 palabras. Se efectúa una operación de memoria cuando la entrada Enable está activa. La entrada ReadWrite determina el tipo de operación. Si ReadWrite es 1, la memoria efectúa una operación de lectura simbolizada por el enunciado

DataOut ← Mem [Address];

Esto hace que se transfieran cuatro bits de la palabra de memoria seleccionada, especificada por Address, a las líneas de salida de datos. Si ReadWrite es 0, la memoria efectúa una operación de escritura simbolizada por el enunciado

Mem [Address] ← DataIn;

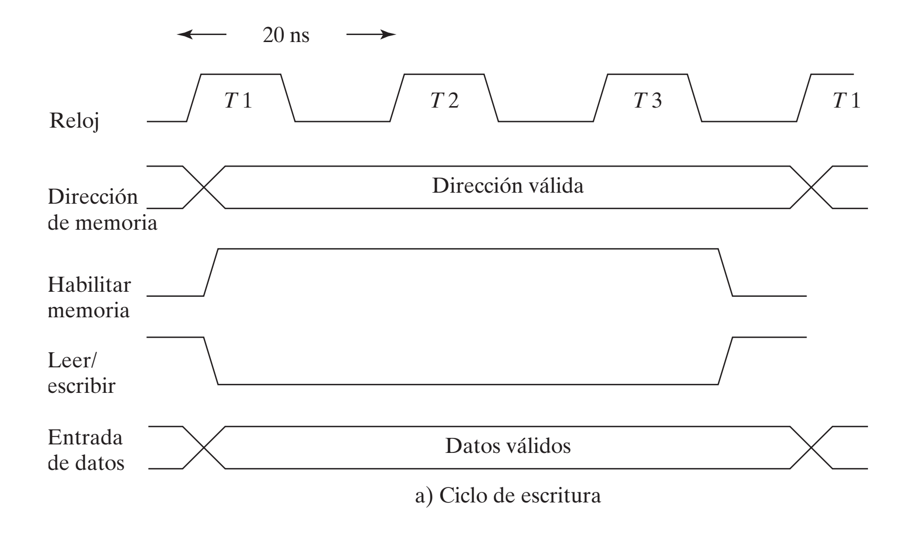
Esto hace que se transfieran cuatro bits de las líneas de entrada de datos a la palabra de memoria seleccionada por Address. Cuando Enable es 0, la memoria queda inhabilitada y se supone que las salidas están en un estado de alta impedancia. Esto se simboliza con la palabra clave **z,** e indica que la memoria tiene salidas de tres estados.

Formas de onda de temporización

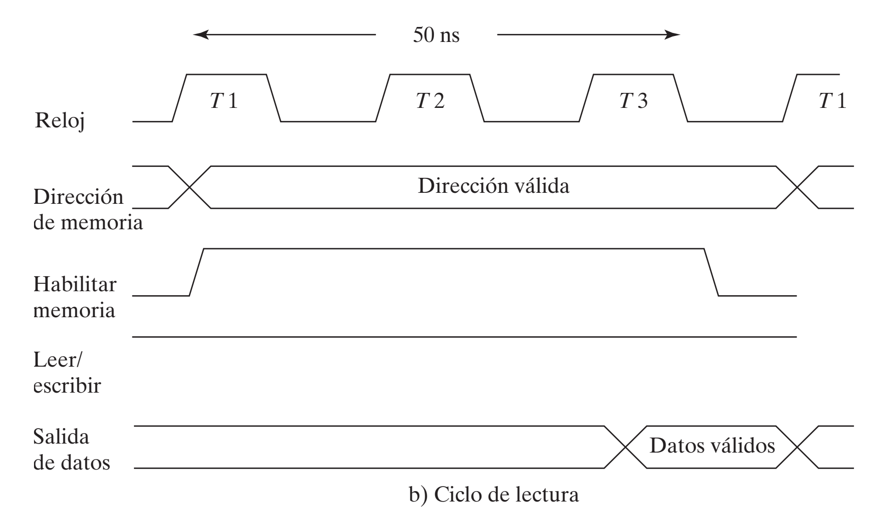
El funcionamiento de la unidad de memoria se controla con un dispositivo externo, como una unidad central de procesamiento (CPU). La CPU suele sincronizarse con su propio reloj. La memoria, en cambio, no utiliza un reloj interno; sus operaciones de lectura y escritura se especifican con entradas de control. El *tiempo de acceso* de una memoria es el tiempo que toma seleccionar una palabra y leerla. El *tiempo de ciclo* de una memoria es el tiempo necesario para llevar a cabo una operación de escritura. La CPU debe suministrar las señales de control de memoria de manera tal que sincronice sus operaciones internas, controladas por reloj, con las operaciones de lectura y escritura de la memoria. Esto implica que el tiempo de acceso y de ciclo de la memoria deben equivaler a un número fijo de ciclos de reloj.

Supongamos, por ejemplo, que una CPU opera con una frecuencia de reloj de 50 MHz, lo que da un periodo de 20 ns para cada ciclo de reloj. Suponga que esta CPU se comunica con una memoria cuyos tiempos de acceso y de ciclo no exceden los 50 ns. Esto implica que el ciclo de escritura termina de guardar la palabra seleccionada en 50 ns o menos, y que el ciclo de lectura proporciona el contenido de la palabra seleccionada en 50 ns o menos. (Los dos tiempos no siempre son iguales.) Puesto que el periodo de la CPU es de 20 ns, será necesario dedicar por lo menos dos y medio, y posiblemente tres, ciclos de reloj a cada solicitud de memoria.

La temporización de memoria que se indica en la figura 7-4 corresponde a una CPU con un reloj de 50 MHz y una memoria con un tiempo máximo de ciclo de 50 ns. El ciclo de escritura de la parte a) muestra tres ciclos de 20 ns: $T1$, $T2$ y $T3$. En una operación de escritura, la CPU deberá suministrar la dirección y los datos de entrada a la memoria. Esto se hace al principio de $T1$. (Las dos líneas que se cruzan en las formas de onda de dirección y datos indican un posible cambio de valor de las líneas múltiples.) Las señales de habilitar memoria y leer/escribir se deben activar una vez que las señales de las líneas de dirección alcanzan la estabilidad, para no destruir los datos almacenados en otra palabra de memoria. La señal de habilitar memoria cambia al nivel alto y la señal de leer/escribir cambia al nivel bajo, para indicar una operación de escritura. Las dos señales de control deben mantenerse activas durante por lo menos 50 ns. Las señales de dirección y de datos deben permanecer estables durante un tiempo corto después de que se desactivan las señales de control. Al término del tercer ciclo de reloj, la operación de escritura en memoria se ha llevado a cabo y la CPU puede tener acceso otra vez a la memoria con el siguiente ciclo $T1$.



a) Ciclo de escritura



b) Ciclo de lectura

FIGURA 7-4
Formas de onda de temporización de un ciclo de memoria

El ciclo de lectura que se aprecia en la figura 7-4b) recibe de la CPU una dirección de memoria. Las señales de habilitar memoria y leer/escribir deben estar en su nivel alto para una operación de lectura. La memoria coloca en las líneas de salida de datos los datos contenidos en la palabra seleccionada en un intervalo de 50 ns (o menos), contados a partir de la activación de la habilitación de memoria. La CPU puede transferir los datos a uno de sus registros internos durante la transición negativa de *T*3. En el siguiente ciclo *T*1 es posible efectuar otra solicitud de memoria.

Tipos de memorias

El modo de acceso de un sistema de memoria depende del tipo de componentes empleados. En una memoria de acceso aleatorio, se considera que las direcciones de palabra están separadas en el espacio, y cada palabra ocupa un lugar dado. En una memoria de acceso secuencial, la información almacenada en algún medio no está accesible inmediatamente; sólo se obtiene en ciertos intervalos de tiempo. Los discos magnéticos o unidades de cinta son de este tipo. Cada posición de memoria pasa por las cabezas de lectura/escritura en cierto orden, pero sólo se lee información cuando se ha llegado a la palabra solicitada. En una memoria de acceso aleatorio, el tiempo de acceso siempre es el mismo, sin importar en qué lugar esté la palabra. En una memoria de acceso secuencial, el tiempo que toma tener acceso a una palabra depende de la posición de la palabra con respecto a la posición de la cabeza lectora; por tanto, el tiempo de acceso es variable.

Las unidades de RAM en circuitos integrados tienen uno de dos modos de operación: *estático* y *dinámico*. La RAM estática (SRAM) consiste básicamente en latches internos que guardan la información binaria. La información binaria será válida en tanto no deje de alimentarse electricidad a la unidad. La RAM dinámica (DRAM) almacena la información binaria en forma de cargas eléctricas en condensadores. Éstos se forman dentro del chip con transistores MOS. La carga almacenada en los condensadores tiende a disiparse con el tiempo, por lo que los condensadores deben recargarse periódicamente *refrescando* la memoria dinámica. El refresco se efectúa restaurando de forma cíclica la carga de todas las palabras cada cierto número de milisegundos, en orden. La DRAM consume menos electricidad y por ello ofrece mayor capacidad de almacenamiento en un solo chip de memoria. La SRAM es más fácil de usar y sus ciclos de lectura y escritura son más cortos.

Las unidades de memoria que pierden la información almacenada cuando se interrumpe la alimentación eléctrica se denominan *volátiles*. Las RAM de circuitos integrados, tanto estáticas como dinámicas, pertenecen a esta categoría, pues las celdas binarias necesitan electricidad externa para mantener la información almacenada. En contraste, una memoria no volátil, como un disco magnético, conserva la información almacenada después de apagarse. Ello se debe a que los datos almacenados en componentes magnéticos están representados por la dirección de imantación, que se mantiene cuando se deja de alimentar electricidad. Otra memoria no volátil es la memoria de sólo lectura (ROM). Se requiere una memoria no volátil en las computadoras digitales para almacenar los programas que se necesitarán la próxima vez que se encienda la computadora después de apagarse. Los programas y datos que no pueden alterarse se guardan en ROM. Otros programas grandes se mantienen en discos magnéticos. Cuando se enciende la computadora, puede usar los programas que están en ROM. Luego, los demás programas que residen en disco magnético se transfieren a la RAM de la computadora, conforme se vayan necesitando. Antes de apagar la computadora, la información binaria contenida en su RAM y que se desea conservar se transfiere al disco.

7-3 DECODIFICACIÓN DE MEMORIA

Además de los componentes de almacenamiento de la unidad de memoria, se requieren circuitos de decodificación para seleccionar la palabra de memoria especificada por las líneas de dirección. En esta sección se presentará la construcción interna de una memoria de acceso aleatorio y se ilustrará el funcionamiento del decodificador. Para poder incluir toda la memoria en un diagrama, la unidad de memoria que usaremos como ejemplo sólo tiene capacidad para 16 bits dispuestos en cuatro palabras de cuatro bits cada una. Se incluye un ejemplo de decodificación coincidente bidimensional para ilustrar un esquema de decodificación más eficiente que se usa en las memorias grandes. Luego mostraremos un ejemplo de multiplexión de direcciones que se usa comúnmente en los circuitos integrados de DRAM.

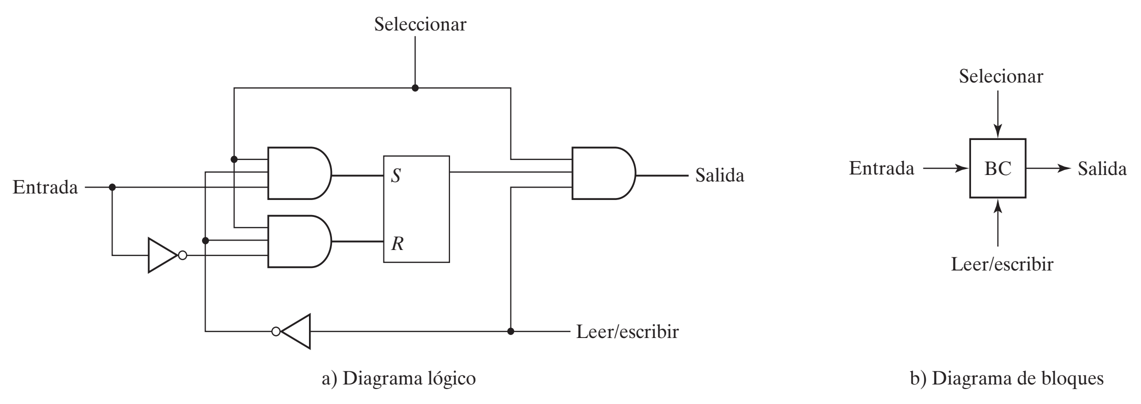


FIGURA 7-5
Celda de memoria

Construcción interna

La construcción interna de una memoria de acceso aleatorio de m palabras y n bits por palabra consta de $m \times n$ celdas binarias de almacenamiento y los correspondientes circuitos de decodificación que seleccionan palabras individuales. La celda binaria de almacenamiento es el bloque de construcción básico de una unidad de memoria. En la figura 7-5 se muestra la lógica equivalente de una celda binaria que guarda un bit de información. La parte de almacenamiento de la celda se modela con un latch SR y algunas compuertas. En realidad, la celda es un circuito electrónico con entre cuatro y seis transistores. No obstante, es posible y conveniente modelarla empleando símbolos lógicos. Las celdas binarias de almacenamiento deben ser muy pequeñas para que muchas de ellas quepan en la reducida área del chip de circuitos integrados. La celda binaria almacena un bit en su latch interno. La entrada de selección habilita a la celda para leer o escribir, y la entrada leer/escribir determina la operación de la celda. Un 1 en esa entrada hace que se efectúe la operación de lectura porque establece una trayectoria entre el latch y la terminal de salida. Un 0 en la entrada de leer/escribir hace que se efectúe la operación de escritura porque establece una trayectoria entre la terminal de entrada y el latch.

En la figura 7-6 se ilustra la construcción lógica de una RAM pequeña. Consta de cuatro palabras de cuatro bits cada una y tiene un total de 16 celdas binarias. Los pequeños bloques rotulados CB representan las celdas binarias con sus tres entradas y una salida, como se especifica en la figura 7-5b). Una memoria de cuatro palabras necesita dos líneas de dirección. Las dos entradas de dirección pasan por un decodificador de 2×4 para seleccionar una de las cuatro palabras. El decodificador se habilita con la entrada de habilitar memoria. Si esa señal es 0, todas las salidas del decodificador son 0 y no se selecciona ninguna de las palabras de memoria. Si la señal es 1, se selecciona una de las cuatro palabras, especificada por el valor de las dos líneas de dirección. Una vez seleccionada una palabra, la entrada leer/escribir determina la operación. Durante la operación de lectura, los cuatro bits de la palabra seleccionada pasan por compuertas OR a las terminales de salida. (Observe que las compuertas OR se dibujaron como el arreglo lógico establecido en la figura 7-1.) Durante la operación de escritura, los datos disponibles en las líneas de entrada se transfieren a las cuatro celdas binarias de la palabra seleccionada. Las celdas binarias que no están seleccionadas se inhabilitan, y sus valores binarios anteriores no cambian. Si la entrada de selección de memoria que llega al decodificador es 0, no se selecciona ninguna de las palabras y el contenido de todas las celdas permanece inalterado, sea cual sea el valor de la entrada leer/escribir.

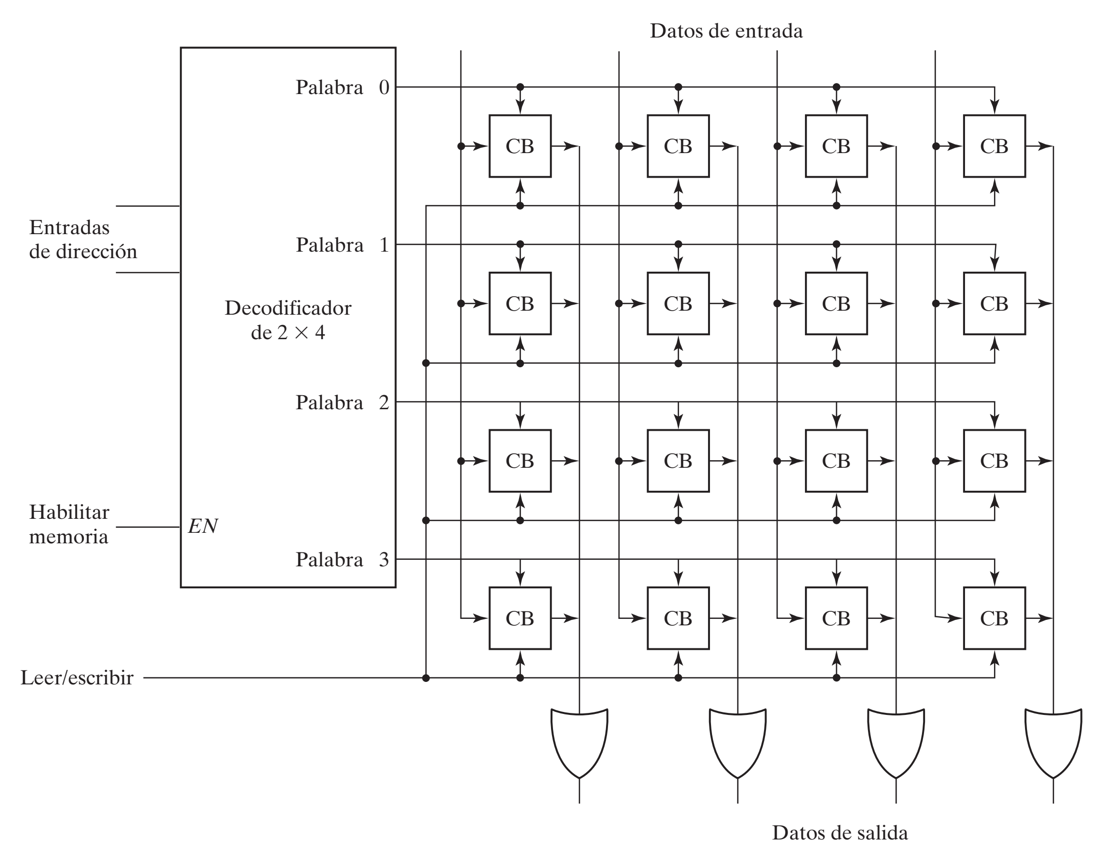


FIGURA 7-6
Diagrama de una RAM de 4 × 6

Las memorias de acceso aleatorio comerciales podrían tener una capacidad de miles de palabras, cada una de las cuales puede tener de 1 a 64 bits. La construcción lógica de una memoria de alta capacidad sería una extensión directa de la configuración que aquí se presenta. Una memoria de 2^k palabras de n bits cada una requiere k líneas de dirección que entran en un decodificador de $k \times 2^k$. Cada salida del decodificador selecciona una palabra de n bits para leerla o escribirla.

Decodificación coincidente

Un decodificador con k entradas y 2^k salidas requiere 2^k compuertas AND con k entradas por compuerta. El total de compuertas y el número de entradas por compuerta se reduce utilizando dos decodificadores en un esquema bidimensional de selección. La idea básica de la decodificación bidimensional es acomodar las celdas de memoria en un arreglo lo más cercano posible a un cuadrado. En esta configuración, se usan dos decodificadores con $k/2$ entradas cada uno, en vez de un decodificador con k entradas. Un decodificador selecciona la fila y el otro selecciona la columna de una configuración de matriz bidimensional.

En la figura 7-7 se ilustra el patrón bidimensional de selección para una memoria de 1K palabras. En lugar de usar un solo decodificador de 10 × 1024, se emplean dos decodificadores

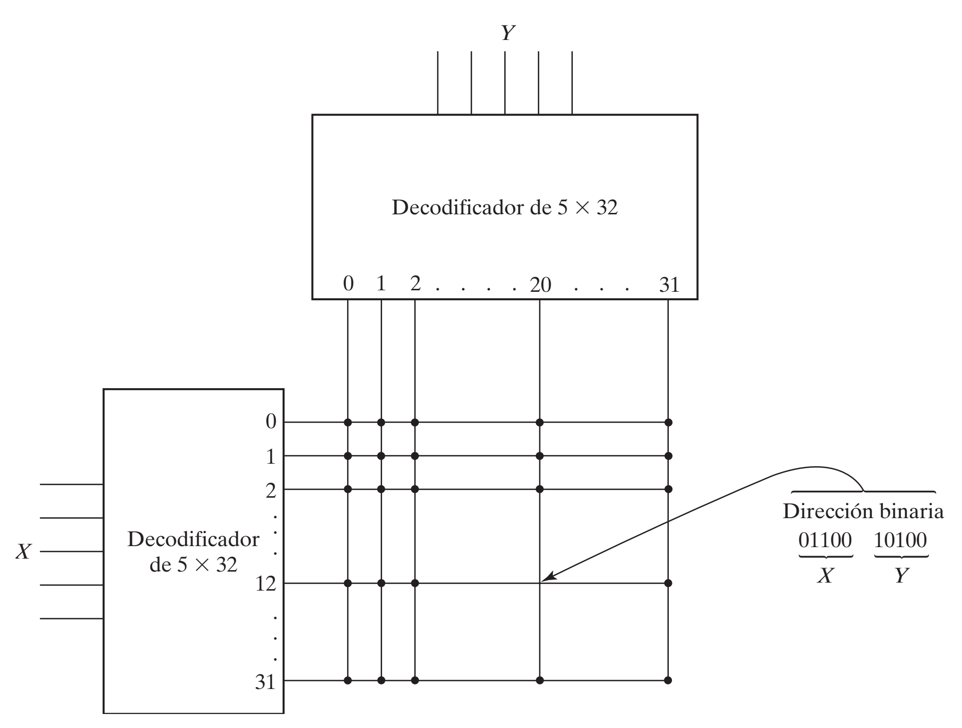


FIGURA 7-7
Estructura bidimensional de decodificación para una memoria de 1K palabras

de 5 × 32. Con el decodificador único necesitaríamos 1024 compuertas AND con 10 entradas cada una. Con los dos decodificadores, se necesitan 64 compuertas AND con cinco entradas cada una. Los cinco bits más significativos de la dirección se envían a la entrada *X*, y los cinco menos significativos, a *Y*. Cada palabra del arreglo de memoria se selecciona por la coincidencia de una línea *X* y una línea *Y*. Así, cada palabra de la memoria se selecciona por la coincidencia de una de 32 filas y una de 32 columnas, para un total de 1024 palabras. Cada intersección representa una palabra que podría tener cualquier cantidad de bits.

Considere, como ejemplo, la palabra cuya dirección es 404. El equivalente binario de 404 es 01100 10100 (con 10 bits). Esto hace que $X = 01100$ (12 binario) y $Y = 10100$ (20 binario). La palabra de *n* bits que se selecciona está en la salida 12 del decodificador *X* y la salida 20 del decodificador *Y*. Se seleccionan para lectura o escritura todos los bits de la palabra.

Multiplexión de direcciones

La celda de memoria SRAM modelada en la figura 7-5 por lo regular contiene seis transistores. Para construir memorias con una mayor densidad es preciso reducir el número de transistores en cada celda. La celda DRAM contiene un transistor MOS y un condensador. La carga almacenada en el condensador se disipa con el tiempo, por lo que las celdas de memoria se deben recargar periódicamente refrescando la memoria. Por lo sencillo de la estructura de sus celdas, las DRAM suelen tener una capacidad cuatro veces mayor que la SRAM. Esto permite incluir cuatro veces más memoria en un chip de un tamaño dado. El costo por bit de memoria DRAM es de tres a cuatro veces menos que el de SRAM. Se logra un ahorro adicional porque

las celdas DRAM consumen menos electricidad. Estas ventajas hacen que la DRAM sea la tecnología preferida para memorias grandes. Hay chips de DRAM con capacidades de 64K bits a 256M bits. Casi todas las DRAM tienen un tamaño de palabra de 1 bit, por lo que es preciso combinar varios chips para tener palabras más grandes.

Por su gran capacidad, la decodificación de direcciones en las DRAM se efectúa con un arreglo bidimensional, y las memorias más grandes suelen tener múltiples areglos. A fin de reducir el número de terminales del paquete de CI, los diseñadores utilizan multiplexión de direcciones, en la que un juego de terminales de entrada da cabida a los componentes de la dirección. En un arreglo bidimensional, la dirección se aplica en dos partes, primero la de fila y luego la de columna. Puesto que se usa el mismo juego de terminales para ambas partes de la dirección, el tamaño del paquete se reduce considerablemente.

Usaremos una memoria de 64K palabras para ilustrar la multiplexión de direcciones. En la figura 7-8 aparece un diagrama de la configuración de decodificación. La memoria consiste en

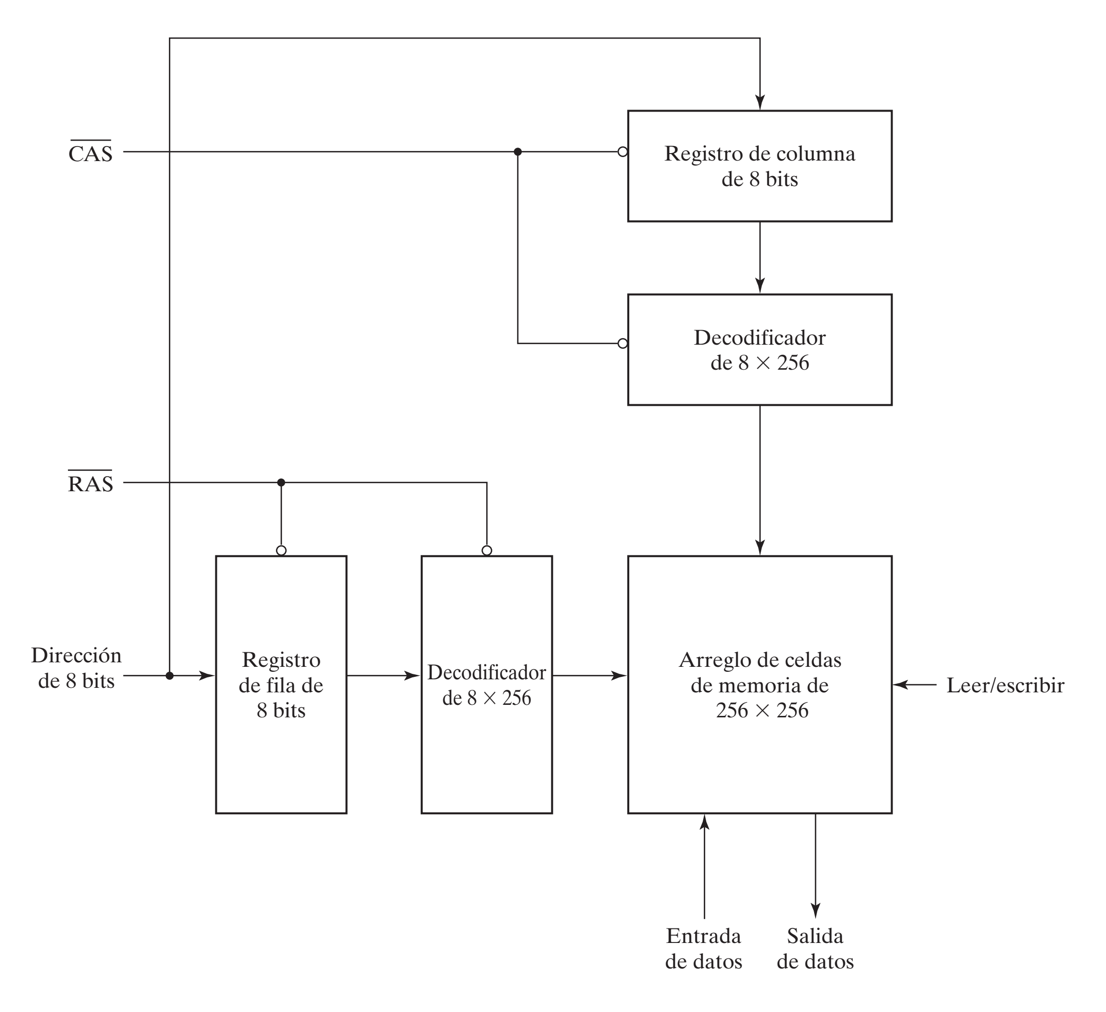


FIGURA 7-8
Multiplexión de direcciones para una DRAM de 64K

un arreglo bidimensional de celdas dispuestas en 256 filas y 256 columnas, para un total de $2^8 \times 2^8 = 2^{16} = 64K$ palabras. Hay una sola línea de entrada de datos, una sola de salida de datos y un control de leer/escribir. Hay una entrada de dirección de ocho bits y dos *strobes* (señales estroboscópicas) de dirección que habilitan las direcciones de fila y de columna para que ingresen en sus respectivos registros. El *strobe* de dirección de fila RAS (*row address strobe*) habilita el registro de fila de ocho bits, y el de columna, CAS (*column address strobe*), habilita el registro de columna de ocho bits. La raya sobre el símbolo de *strobe* indica que los registros se habilitan en en nivel 0 de la señal.

La dirección de 16 bits se aplica a la DRAM en dos pasos, utilizando RAS y CAS. En un principio, ambos *strobes* están en el estado 1. La dirección de fila, de ocho bits, se aplica a las entradas de dirección, y se cambia RAS a 0. Esto carga la dirección de fila en el registro de dirección de fila. RAS también habilita el decodificador de fila para decodificar la dirección de fila y seleccionar una fila del arreglo. Después de un tiempo equivalente al tiempo de estabilización de la selección de fila, RAS vuelve al nivel 1. Entonces se aplica la dirección de columna, de ocho bits, a las entradas de dirección, y CAS se cambia al estado 0. Esto transfiere la dirección de columna al registro de columna y habilita el decodificador de columna. En este momento, las dos partes de la dirección ya están en sus respectivos registros, los decodificadores las han decodificado para seleccionar la celda única correspondiente a las direcciones de fila y de columna, y ya puede efectuarse una operación de lectura/escritura con esa celda. CAS deberá volver al nivel 1 antes de iniciar otra operación de memoria.

7-4 DETECCIÓN Y CORRECCIÓN DE ERRORES

El nivel de complejidad de un arreglo de memoria podría causar errores ocasionales al almacenar y recuperar la información binaria. La fiabilidad de una unidad de memoria aumenta si se usan códigos para detectar y corregir errores. El esquema más común para detectar errores es el bit de paridad (véase la sección 3-8). Se genera un bit de paridad y se almacena junto con la palabra de datos en la memoria. Después de leerse una palabra de la memoria, se verifica su paridad, y se acepta si la paridad de los bits leídos es correcta. Si la verificación de paridad da una inversión, se habrá detectado un error, pero no podrá corregirse.

Un código de corrección de errores genera múltiples bits de comprobación de paridad que se almacenan junto con la palabra de datos en la memoria. Cada bit de comprobación completa la paridad de un grupo de bits de la palabra de datos. Cuando la palabra se lee de la memoria, también se leen los bits de paridad y se comparan con un nuevo juego de bits de comprobación generados a partir de los datos leídos. Si los bits de comprobación coinciden, quiere decir que no hubo error; si no coinciden, generan un patrón único, llamado *síndrome*, que sirve para identificar el bit erróneo. Hay un error individual cuando un bit cambia de 1 a 0 o de 0 a 1 durante la operación de escritura o lectura. Si se identifica el bit erróneo, el error podrá corregirse complementando dicho bit.

Código Hamming

Uno de los códigos para corrección de errores más utilizados en memorias de acceso aleatorio fue ideado por R. W. Hamming. En el código Hamming, se añaden k bits de paridad a una palabra de datos de n bits para formar una nueva palabra de $n + k$ bits. Las posiciones de bit se numeran sucesivamente de 1 a $n + k$. Las posiciones numeradas con potencias de 2 se re-

servan para los bits de paridad. Los demás bits son los bits de datos. Este código puede usarse con palabras de cualquier longitud. Antes de presentar las características generales del código, ilustraremos su funcionamiento con una palabra de datos de ocho bits.

Consideremos, por ejemplo, la palabra de datos 11000100. Incluimos cuatro bits de paridad con la palabra de ocho bits y acomodamos los 12 bits como sigue:

Posición de bit:	1	2	3	4	5	6	7	8	9	10	11	12
	P_1	P_2	1	P_4	1	0	0	P_8	0	1	0	0

Los cuatro bits de paridad, P_1, P_2, P_4 y P_8, están en las posiciones 1, 2, 4 y 8, respectivamente. Los ocho bits de la palabra de datos están en las posiciones restantes. Cada bit de paridad se calcula como sigue:

$$P_1 = \text{XOR de los bits } (3, 5, 7, 9, 11) = 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 = 0$$
$$P_2 = \text{XOR de los bits } (3, 6, 7, 10, 11) = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0$$
$$P_4 = \text{XOR de los bits } (5, 6, 7, 12) = 1 \oplus 0 \oplus 0 \oplus 0 = 1$$
$$P_8 = \text{XOR de los bits } (9, 10, 11, 12) = 0 \oplus 1 \oplus 0 \oplus 0 = 1$$

Recuerde que la operación OR exclusivo efectúa la función impar. Es 1 si hay un número impar de unos en las variables, y 0 si hay un número par de unos. Así, cada bit de paridad se ajusta de modo que el número total de unos en las posiciones verificadas, incluido el bit de paridad, siempre sea par.

La palabra de datos de ocho bits se almacena en la memoria junto con los cuatro bits de paridad como una palabra compuesta de 12 bits. Si colocamos los cuatro bits P en la posición que les corresponde, obtendremos la palabra compuesta que se guarda en la memoria:

	0	0	1	1	1	0	0	1	0	1	0	0
Posición de bit:	1	2	3	4	5	6	7	8	9	10	11	12

Cuando los 12 bits se leen de la memoria, se revisan para ver si hubo errores. La paridad se verifica para las mismas combinaciones de bits que incluyen el bit de paridad. Los cuatro bits de comprobación se evalúan así:

$$C_1 = \text{XOR de los bits } (1, 3, 5, 7, 9, 11)$$
$$C_2 = \text{XOR de los bits } (2, 3, 6, 7, 10, 11)$$
$$C_4 = \text{XOR de los bits } (4, 5, 6, 7, 12)$$
$$C_8 = \text{XOR de los bits } (8, 9, 10, 11, 12)$$

Un bit de comprobación 0 indica que los bits verificados tienen paridad par, y un 1, que tienen paridad impar. Puesto que los bits se almacenaron con paridad par, el resultado C $= C_8C_4C_2C_1 = 0000$ indica que no hubo error. En cambio, si $C \neq 0$, el número binario de cuatro bits formado por los bits de comprobación da la posición del bit erróneo. Por ejemplo, consideremos estos tres casos:

Posición de bit:	1	2	3	4	5	6	7	8	9	10	11	12	
	0	0	1	1	1	0	0	1	0	1	0	0	No hubo error
	1	0	1	1	1	0	0	1	0	1	0	0	Error en el bit 1
	0	0	1	1	0	0	0	1	0	1	0	0	Error en el bit 5

En el primer caso, no hay errores en la palabra de 12 bits. En el segundo, hay un error en la posición de bit 1 porque cambió de 0 a 1. El tercer caso muestra un error en la posición de bit 5 que cambió de 1 a 0. Al evaluar el XOR de los bits correspondientes, obtenemos estos bits de comprobación:

	C_8	C_4	C_2	C_1
Sin error:	0	0	0	0
Con error en el bit 1:	0	0	0	1
Con error en el bit 5:	0	1	0	1

Así, cuando no hay error, tenemos $C = 0000$; con un error en el bit 1, obtenemos $C = 0001$; y con un error en el bit 5, obtenemos $C = 0101$. El valor binario de C, si es distinto de 0000, da la posición del bit erróneo. El error se corrige complementando el bit correspondiente. Cabe señalar que el error podría presentarse en la palabra de datos o en uno de los bits de paridad.

El código Hamming se puede utilizar con palabras de datos de cualquier longitud. En general, el código consiste en k bits de comprobación y n bits de datos, para dar un total de $n + k$ bits. El valor del síndrome C tiene k bits y abarca 2^k valores entre 0 y $2^k - 1$. Uno de estos valores, que por lo regular es el cero, indica que no se detectó ningún error, así que quedan $2^k - 1$ valores para indicar cuál de los $n + k$ bits estaba equivocado. Cada uno de estos $2^k - 1$ valores para describir de forma inconfundible un bit erróneo. Por tanto, el intervalo de k debe ser mayor o igual que $n + k$, lo que da la relación

$$2^k - 1 \geq n + k$$

Al despejar n en términos de k, se obtiene

$$2^k - 1 - k \geq n$$

Esta relación da una fórmula para establecer el número de bits de datos que se pueden usar junto con k bits de comprobación. Por ejemplo, cuando $k = 3$, el número de bits de datos que se pueden usar es $n \leq (2^3 - 1 - 3) = 4$. Con $k = 4$, tenemos $2^4 - 1 - 4 = 11$, o sea que $n \leq 11$. La palabra de datos puede tener menos de 11 bits, pero debe tener por lo menos cinco bits, pues con menos de cinco bits sólo se necesitan tres bits de comprobación. Esto justifica el uso de cuatro bits de comprobación para los ocho bits de datos del ejemplo anterior. En la tabla 7-2 se dan los intervalos de n para diversos valores de k.

La agrupación de bits para generar y verificar la paridad se obtiene de una lista de los números binarios de 0 hasta $2^k - 1$. El bit menos significativo es 1 en los números binarios 1, 3,

Tabla 7-2
Intervalo de bits de datos para k bits de comprobación

Número de bits de comprobación, k	Intervalo de bits de datos, n
3	2-4
4	5-11
5	12-26
6	27-57
7	58-120

5, 7, etcétera. El segundo bit significativo es 1 en los números binarios 2, 3, 6, 7, etcétera. Si comparamos estos números con las posiciones de bit que se usan para generar y verificar bits de paridad en el código Hamming, notaremos la relación entre la agrupación de bits en el código y la posición de los bits 1 en la sucesión de conteo binario. Cada grupo de bits inicia con una potencia de 2, como 1, 2, 4, 8, 16, etcétera. Estos números son también los números de posición de los bits de paridad.

Corrección de errores individuales, detección de errores dobles

El código Hamming sólo puede detectar y corregir un error. No se detectan múltiples errores. Si añadimos otro bit de paridad a la palabra codificada, será posible usar el código Hamming para corregir un solo error y detectar errores dobles. Si incluimos este bit adicional de paridad, la palabra que antes codificamos con 12 bits se convierte en 001110010100P_{13}, donde P_{13} se evalúa efectuando el OR exclusivo de los otros 12 bits. Esto produce la palabra de 13 bits 0011100101001 (paridad par). Cuando se lee de la memoria la palabra de 13 bits, se evalúan los bits de comprobación y también la paridad P de los 13 bits. Si $P = 0$, quiere decir que la paridad es correcta (paridad par), pero si $P = 1$, la paridad de los 13 bits es incorrecta (paridad impar). Se pueden presentar cuatro casos:

Si $C = 0$ y $P = 0$, no hubo error.

Si $C \neq 0$ y $P = 1$, hubo un solo error que puede corregirse.

Si $C \neq 0$ y $P = 0$, hubo un doble error que se detecta pero que no es posible corregir.

Si $C = 0$ y $P = 1$, hubo un error en el bit P_{13}.

Este esquema podría detectar más de dos errores, pero no garantiza la detección de todos esos errores.

Los circuitos integrados usan un código Hamming modificado para generar y verificar bits de paridad que permiten corregir errores individuales y detectar errores dobles. El código Hamming modificado usa una configuración de paridad más eficiente que balancea el número de bits con los que se calcula el XOR. Un CI que usa una palabra de datos de ocho bits y una palabra de comprobación de cinco bits es el tipo 74637. Hay otros circuitos integrados para palabras de datos de 16 y 32 bits. Estos circuitos se utilizan junto con una unidad de memoria para corregir errores individuales o detectar errores dobles durante las operaciones de escribir y leer.

7-5 MEMORIA DE SÓLO LECTURA

Una memoria de sólo lectura (ROM) es, en esencia, un dispositivo de memoria en el que se almacena información binaria permanente. El diseñador debe especificar la información, que entonces se incorpora a la unidad para formar el patrón de interconexión requerido. Una vez establecido el patrón, permanece en la unidad aunque se apague y se vuelva a encender.

En la figura 7-9 se reproduce un diagrama de bloques de una ROM. Tiene k entradas y n salidas. Las entradas proporcionan la dirección de memoria y las salidas suministran los bits de datos de la palabra almacenada seleccionada por la dirección. El número de palabras de una ROM está determinado por el hecho de que se necesitan k líneas de dirección para especificar

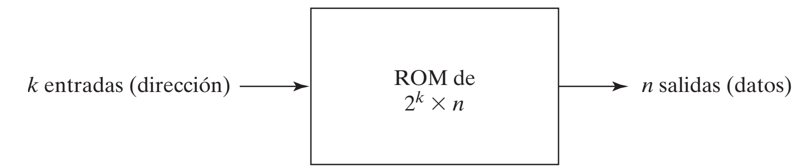


FIGURA 7-9
Diagrama de bloques de ROM

2^k palabras. La ROM no tiene entradas de datos porque no efectúa la operación de escritura. Los chips de circuitos integrados de ROM tienen una o más entradas de habilitación y a veces cuentan con salidas de tres estados que facilitan la construcción de grandes arreglos de ROM.

Consideremos, por ejemplo, una ROM de 32×8. La unidad consiste en 32 palabras de 8 bits cada una. Hay cinco líneas de entrada que forman los números binarios del 0 al 31 para la dirección. La figura 7-10 muestra la construcción lógica interna de la ROM. Las cinco entradas se decodifican a 32 salidas distintas con un decodificador de 5×32. Cada salida del decodificador representa una dirección de memoria. Las 32 salidas del decodificador se conectan a cada una de las ocho compuertas OR. El diagrama muestra la convención de arreglos lógicos que se emplea en circuitos complejos (véase la figura 7-1). Debe considerarse que cada compuerta OR tiene 32 entradas. Cada salida del decodificador se conecta a una de las entradas de cada compuerta OR. Puesto que cada compuerta OR tiene 32 conexiones de entrada y hay ocho compuertas OR, la ROM contiene $32 \times 8 = 256$ conexiones internas. En general, una ROM de $2^k \times n$ tiene un decodificador interno de $k \times 2^k$ y n compuertas OR. Cada compuerta OR tiene 2^k entradas, que se conectan a cada una de las salidas del decodificador.

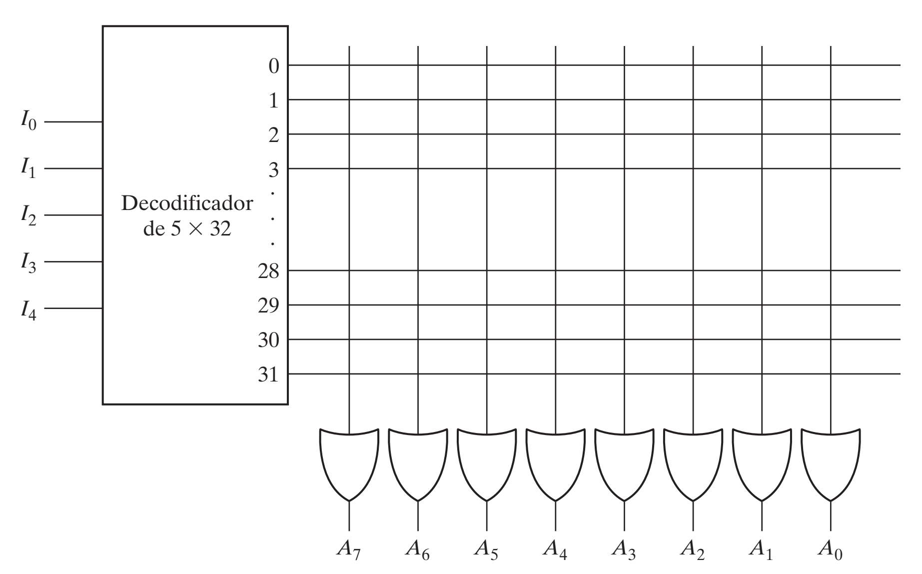


FIGURA 7-10
Lógica interna de una ROM de 32×8

Tabla 7-3
Tabla de verdad de ROM (parcial)

Entradas					Salidas							
I4	**I3**	**I2**	**I1**	**I0**	**A7**	**A6**	**A5**	**A4**	**A3**	**A2**	**A1**	**A0**
0	0	0	0	0	1	0	1	1	0	1	1	0
0	0	0	0	1	0	0	0	1	1	1	0	1
0	0	0	1	0	1	1	0	0	0	1	0	1
0	0	0	1	1	1	0	1	1	0	0	1	0
		⋮					⋮					
1	1	1	0	0	0	0	0	0	1	0	0	1
1	1	1	0	1	1	1	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	1	0	1	0
1	1	1	1	1	0	0	1	1	0	0	1	1

Las 256 interconexiones de la figura 7-10 son programables. Una conexión programable entre dos líneas equivale lógicamente a un interruptor que se puede alterar de modo que esté cerrado (o sea, que las dos líneas están conectadas) o abierto (o sea, que las dos líneas están desconectadas). La intersección programable entre dos líneas se conoce como punto de cruce. Se usan diversos dispositivos físicos para implementar interruptores de punto de cruce. Una de las tecnologías más sencillas utiliza un fusible que normalmente conecta los dos puntos, pero que se abre o "quema" con un pulso de alto voltaje.

El almacenamiento binario interno de una ROM se especifica con una tabla de verdad que indica el contenido de palabra de cada dirección. Por ejemplo, el contenido de una ROM de 32×8 se especifica con una tabla de verdad similar a la que se observa en la tabla 7-3. Esa tabla presenta las cinco entradas bajo las que se da una lista de las 32 direcciones. En cada dirección, está almacenada una palabra de ocho bits, que se da bajo las columnas de salida. La tabla sólo muestra las primeras cuatro y las últimas cuatro palabras de la ROM. La tabla completa debe incluir la lista de las 32 palabras.

El procedimiento en hardware que programa la ROM hace que se quemen fusibles según una tabla de verdad dada. Por ejemplo, la programación de la ROM según la tabla de verdad de la tabla 7-3 produce la configuración que se aprecia en la figura 7-11. Cada 0 de la tabla de verdad especifica una ausencia de conexión, y cada 1, especifica una trayectoria que se obtiene con una conexión. Por ejemplo, la tabla especifica la palabra de ocho bits 10110010 que se almacenará permanentemente en la dirección 3. Los cuatro ceros de la palabra se programan quemando los fusibles entre la salida 3 del decodificador y las entradas de las compuertas OR asociadas a las salidas A_6, A_3, A_2 y A_0. Los cuatro unos de la palabra se han marcado en el diagrama con $\times$ para denotar una conexión, en lugar del punto que se usa para indicar una conexión permanente en los diagramas lógicos. Cuando la entrada de la ROM es 00011, todas las salidas del decodificador son 0 excepto la salida 3, que es 1 lógico. La señal equivalente a 1 lógico en la salida 3 del decodificador se propaga por las conexiones hasta las salidas de compuerta OR A_7, A_5, A_4 y A_1. Las otras cuatro salidas siguen en 0. El resultado es que la palabra almacenada 10110010 se aplica a las ocho salidas de datos.

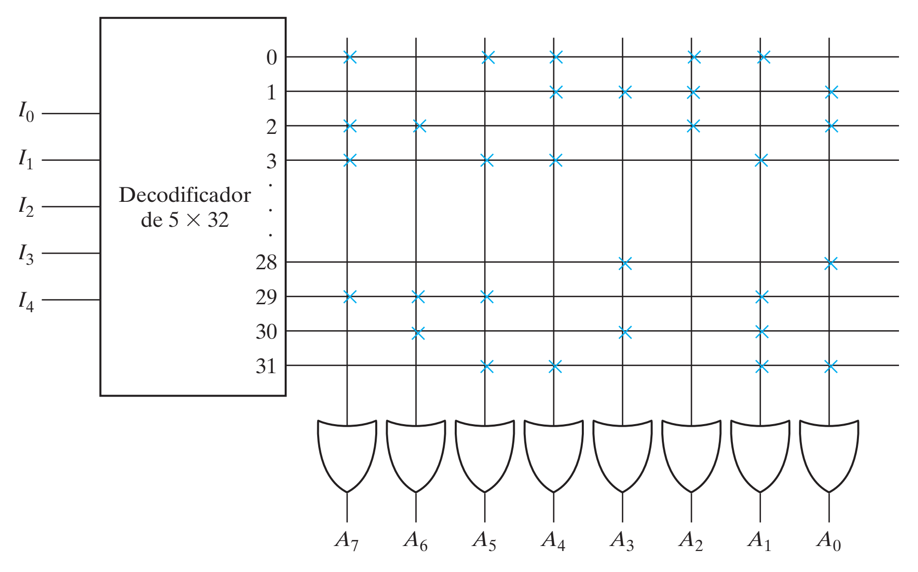


FIGURA 7-11
Programación de la ROM según la tabla 7-3

Implementación de circuitos combinacionales

Ya se explicó en la sección 4-8 que un decodificador genera los 2^k minitérminos de las k variables de entrada. Insertando compuertas OR para sumar los minitérminos de funciones booleanas, era posible generar cualquier circuito combinacional deseado. La ROM es, en esencia, un dispositivo que incluye tanto el decodificador como las compuertas OR dentro de un solo dispositivo. Si escogemos conexiones para los minitérminos incluidos en la función, podremos programar las salidas de la ROM de modo que representen las funciones booleanas de las variables de salida en un circuito combinacional.

El funcionamiento interno de una ROM se interpreta de dos maneras. La primera interpretación es como una unidad de memoria que contiene un patrón fijo de palabras almacenadas. La segunda interpretación es como una unidad que implementa un circuito combinacional. Desde este punto de vista, cada terminal de salida se considera individualmente como salida de una función booleana que se expresa como suma de minitérminos. Por ejemplo, la ROM de la figura 7-11 podría verse como un circuito combinacional que tiene ocho salidas, cada una de las cuales es función de las cinco variables de entrada. La salida A_7 se expresa como suma de minitérminos así:

$$A_7(I_4, I_3, I_2, I_1, I_0) = \Sigma(0, 2, 3, \ldots, 29)$$

(Los puntos suspensivos representan a los minitérminos 4 a 27, que no se especifican en la figura.) Una conexión marcada con × en la figura produce un minitérmino para la suma. Todos los demás puntos de cruce están desconectados y no se incluyen en la suma.

En la práctica, cuando se diseña un circuito combinacional con una ROM, no es necesario diseñar la lógica ni mostrar las conexiones internas de compuertas dentro de la unidad. Lo único que necesita hacer el diseñador es especificar la ROM en cuestión con su número de CI y proporcionar la tabla de verdad de la ROM. Dicha tabla contiene toda la información necesaria para programar la ROM. No se necesita un diagrama de lógica interna además de la tabla.

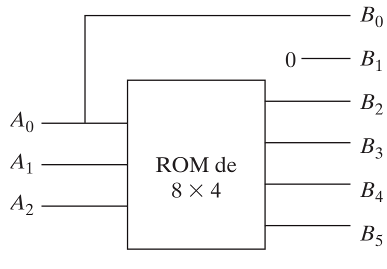
EJEMPLO 7-1

Diseñar un circuito combinacional con una ROM. El circuito acepta un número de tres bits y genera un número binario igual al cuadrado del número introducido.

El primer paso es deducir la tabla de verdad del circuito combinacional. En la mayoría de los casos no se necesita más. En otros casos, se utiliza una tabla de verdad parcial para la ROM aprovechando ciertas propiedades de las variables de salida. La tabla 7-4 es la tabla de verdad del circuito combinacional. Se requieren tres entradas y seis salidas para dar cabida a todos los números binarios posibles. Observamos que la salida B_0 siempre es igual a la entrada A_0, por lo que no hay necesidad de generar B_0 con una ROM, pues es igual a una variable de entrada. Además, la salida B_1 siempre es 0, así que esta salida es una constante conocida. En realidad, sólo necesitamos generar cuatro salidas con la ROM; las otras dos se obtienen fácilmente. La ROM mínima requerida debe tener tres entradas y cuatro salidas. Tres entradas especifican ocho palabras, así que el tamaño de la ROM es 8×4. La implementación en ROM se ilustra en la figura 7-12. Las tres entradas especifican ocho palabras de cuatro bits cada una. La tabla de verdad de la figura 7-12b) especifica la información necesaria para programar la ROM. El diagrama de bloques de la figura 7-12a) indica las conexiones requeridas del circuito combinacional.

Tabla 7-4
Tabla de verdad para el circuito del ejemplo 7-1

Entradas			Salidas						
A_1	A_1	A_0	B_5	B_4	B_3	B_2	B_1	B_0	Decimal
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0	4
0	1	1	0	0	1	0	0	1	9
1	0	0	0	1	0	0	0	0	16
1	0	1	0	1	1	0	0	1	25
1	1	0	1	0	0	1	0	0	36
1	1	1	1	1	0	0	0	1	49



a) Diagrama de bloques

A_2	A_1	A_0	B_5	B_4	B_3	B_2
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	1
0	1	1	0	0	1	0
1	0	0	0	1	0	0
1	0	1	0	1	1	0
1	1	0	1	0	0	1
1	1	1	1	1	0	0

b) Tabla de verdad de la ROM

FIGURA 7-12
Implementación en ROM del ejemplo 7-1

Tipos de ROM

Las trayectorias que se requieren en una ROM se pueden programar de cuatro maneras. La primera se denomina *programación por máscara* y la efectúa el fabricante de semiconductores durante el último proceso de fabricación de la unidad. El procedimiento de manufactura de una ROM requiere que el cliente llene la tabla de verdad que la ROM debe satisfacer. La tabla se presenta en una forma especial proporcionada por el fabricante, o en un formato dado en un medio de salida de computadora. El fabricante crea la máscara correspondiente a las trayectorias que producen los unos y ceros indicados en la tabla de verdad del cliente. Este procedimiento es costoso porque el proveedor cobra al cliente un cargo especial por hacer una máscara a la medida para la ROM en cuestión. Por ello, la programación por máscara sólo resulta económica si el pedido es por grandes cantidades de ROM con la misma configuración.

Para cantidades pequeñas, es más económico usar otro tipo de ROM llamada memoria programable de sólo lectura (PROM, *programmable read-only memory*). Recién compradas, las unidades PROM tienen todos sus fusibles intactos, lo que equivale a 1 en todos los bits de las palabras almacenadas. Los fusibles de la PROM se queman aplicando un pulso de alto voltaje al dispositivo a través de una terminal especial. Un fusible quemado define un estado binario 0 y un fusible intacto da un estado binario 1. Esto permite al usuario programar la PROM en el laboratorio para obtener la relación deseada entre direcciones de entrada y palabras almacenadas. Se venden instrumentos especiales llamados programadores de PROM que facilitan este procedimiento. En cualquier caso, todos los procedimientos para programar las ROM son procedimientos en hardware, aunque se use la palabra programación.

El procedimiento para programar las ROM y PROM por hardware es irreversible y, una vez efectuado, el patrón es permanente y no puede alterarse. Una vez que se ha establecido un patrón de bits, la unidad tendrá que desecharse si es necesario modificar el patrón de bits. Un tercer tipo de ROM es la PROM borrable (EPROM, *erasable* PROM). La EPROM se puede restaurar al estado inicial aunque se le haya programado previamente. Cuando la EPROM se coloca bajo una lámpara ultravioleta especial durante cierto tiempo, la radiación de onda corta descarga las compuertas flotantes internas que actúan como conexiones programadas. Una vez borrada la EPROM vuelve a su estado inicial y es posible reprogramarse con otro conjunto de valores.

El cuarto tipo de ROM es la PROM borrable eléctricamente (EEPROM o E^2PROM). Es como la EPROM, sólo que las conexiones previamente programadas se borran con una señal eléctrica en vez de luz ultravioleta. La ventaja es que el dispositivo puede borrarse sin desmontarlo de su base.

PLD combinacionales

La PROM es un dispositivo lógico programable (PLD, *programmable logic device*) combinacional. Un PLD combinacional es un circuito integrado con compuertas programables divididas en un arreglo AND y un arreglo OR para efectuar una implementación de suma de productos AND-OR. Hay tres tipos principales de PLD combinacionales, que difieren en la colocación de las conexiones programables en el arreglo AND-OR. La figura 7-13 reproduce la configuración de los tres PLD. La memoria programable de sólo lectura (PROM) tiene un arreglo AND fijo construido como decodificador y un arreglo OR programable. Las compuertas OR programables implementan las funciones booleanas como suma de minitérminos. El arreglo lógico programable (PAL, *programmable array logic*) tiene un arreglo AND programable y un arreglo OR

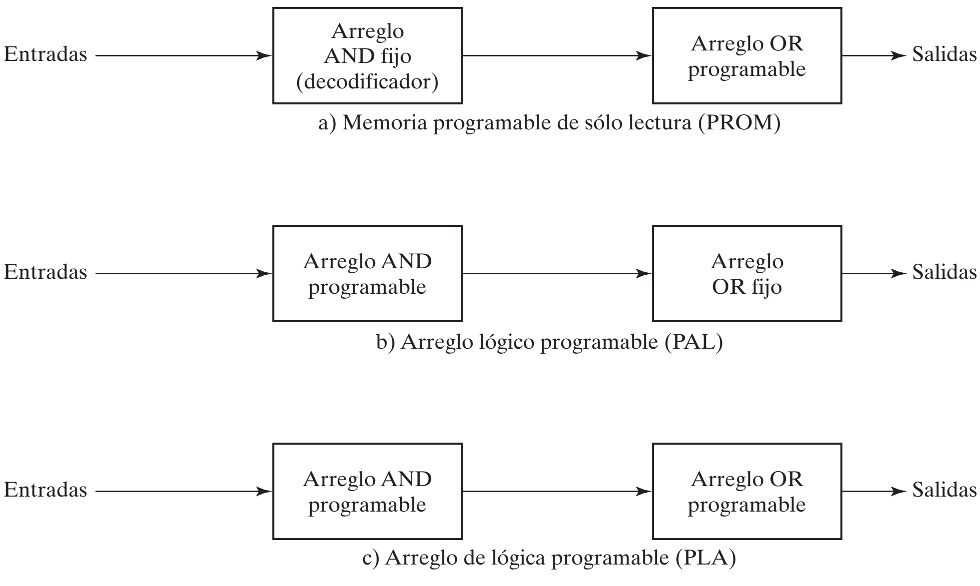


FIGURA 7-13
Configuración básica de tres PLD

fijo. Las compuertas AND se programan para crear los términos producto de las funciones booleanas, que se suman lógicamente en cada compuerta OR. El PLD más flexible es el arreglo de lógica programable (PLA, *programmable logic array*), en el que ambos arreglos, AND y OR, pueden programarse. Cualquier compuerta OR comparte los términos producto del arreglo AND para crear la implementación de suma de productos deseada. Los nombres PAL y PLA surgieron de diferentes fabricantes durante el desarrollo de los dispositivos lógicos programables. En esta sección se ilustra la implementación de circuitos combinacionales con PROM. En las dos secciones que siguen se presentará el diseño de circuitos combinacionales con PLA y PAL.

7-6 ARREGLO DE LÓGICA PROGRAMABLE

El arreglo de lógica programable (PLA) es similar al PROM en su concepto, sólo que el PLA no efectúa una decodificación cabal de las variables ni genera todos los minitérminos. El decodificador se sustituye por un arreglo de compuertas AND que se programan para generar cualquier término producto de las variables de entrada. Luego, los términos producto se conectan a compuertas OR para formar la suma de productos de las funciones booleanas deseadas.

En la figura 7-14 se muestra la lógica interna de un PLA con tres entradas y dos salidas. Todos los circuitos comerciales son más grandes, pero presentamos éste aquí para ilustrar la configuración lógica típica de los PLA. El diagrama utiliza los símbolos gráficos de arreglos lógicos para circuitos complejos. Cada entrada pasa por un búfer y un inversor indicados en el diagrama con un símbolo gráfico compuesto que posee ambas salidas, verdadera y complemento. Cada entrada y su complemento se conectan a las entradas de cada compuerta AND como indican las intersecciones entre las líneas verticales y horizontales. Las salidas de las compuer-

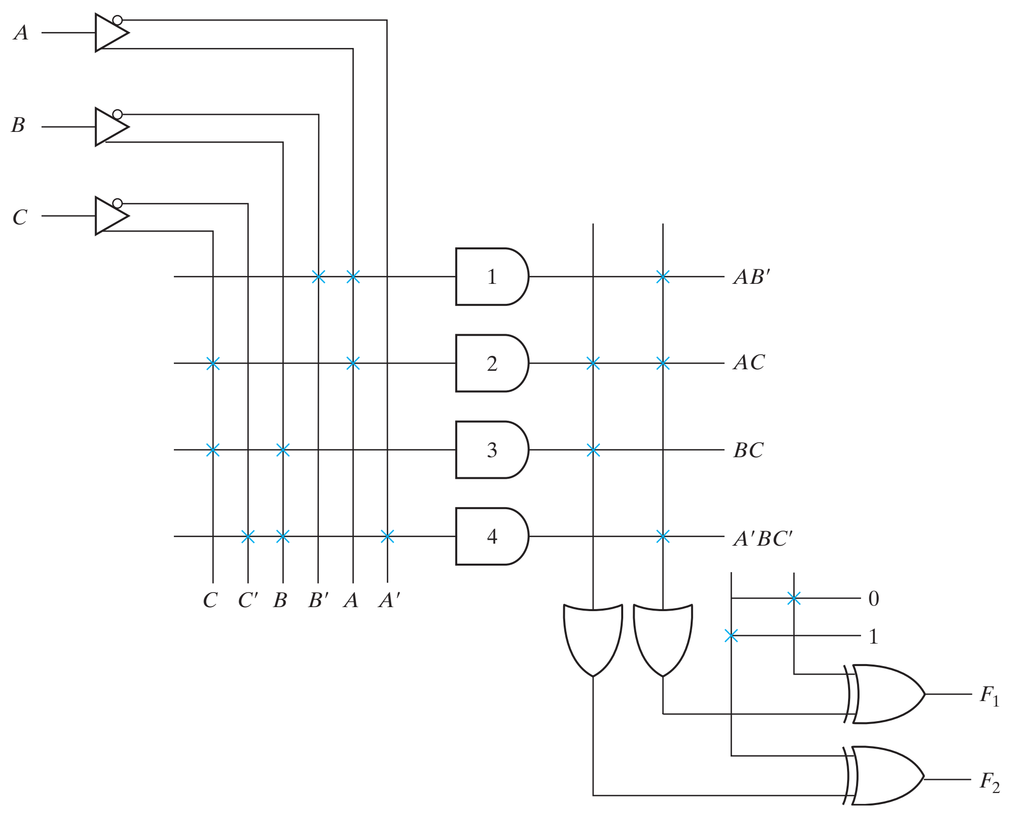


FIGURA 7-14
PLA con tres entradas, cuatro términos producto y dos salidas

tas AND se conectan a las entradas de cada compuerta OR. La salida de la compuerta OR va a una compuerta XOR cuya otra entrada se puede programar de modo que reciba una señal de 1 o 0 lógico. La salida se invierte cuando la entrada XOR se conecta a 1 (porque $x \oplus 1 = x'$). La salida no cambia cuando la entrada XOR se conecta a 0 (porque $x \oplus 0 = x$). Las funciones booleanas implementadas en el PLA de la figura 7-14 son

$$F_1 = AB' + AC + A'BC'$$
$$F_2 = (AC + BC)'$$

Los términos producto generados en cada compuerta AND se dan frente a la salida de la compuerta en el diagrama. El término producto se determina a partir de las entradas cuyos puntos de cruce están conectados y marcados con $\times$. La salida de una compuerta OR da la suma lógica de los términos producto seleccionados. La salida podría complementarse o dejarse en su forma verdadera dependiendo de la conexión de la otra entrada de la compuerta XOR.

El mapa de fusibles de un PLA se especifica en forma tabular. Por ejemplo, en la tabla 7-5 se da la tabla de programación que especifica el PLA de la figura 7-14. La tabla de programación de PLA consta de tres secciones. La primera da una lista numérica de los términos producto. La segunda especifica las trayectorias requeridas entre las entradas y las compuertas AND.

Tabla 7-5
Tabla de programación de PLA

		Entradas			Salidas	
					(V)	(C)
Término producto		A	B	C	F_1	F_2
AB′	1	1	0	–	1	–
AC	2	1	–	1	1	1
BC	3	–	1	1	–	1
A′BC′	4	0	1	0	1	–

La tercera especifica las trayectorias entre las compuertas AND y OR. Para cada variable de salida, podríamos tener una V (de verdadera) o C (de complemento) para programar la compuerta XOR. Los términos producto que se dan a la izquierda no forman parte de la tabla; se han incluido sólo como referencia. Para cada término producto, las entradas se marcan con 1, 0 o – (guión). Si una variable del término producto aparece en su forma verdadera, la variable de entrada correspondiente se marca con 1. Si aparece complementada, la variable de entrada correspondiente se marca con 0. Si la variable no está presente en el término producto, se marca con un guión.

Las trayectorias entre las entradas y las compuertas AND se especifican en las columnas de *entradas* de la tabla de programación. Un 1 en la columna de entrada especifica una conexión de la variable de entrada a la compuerta AND. Un 0 en la columna especifica una conexión entre el complemento de la variable y la entrada de la compuerta AND. Un guión especifica un fusible quemado tanto en la variable de entrada como en su complemento. Se supone que una terminal abierta en la entrada de una compuerta AND se comporta como un 1.

Las trayectorias entre las compuertas AND y OR se especifican en las columnas de *salidas*. Las variables de salida se marcan con 1 para los términos producto que están incluidos en la función. Cada término producto que tiene un 1 en la columna de salida requiere un camino de la salida de la compuerta AND a la entrada de la compuerta OR. Los marcados con un guión especifican un fusible quemado. Se supone que una terminal abierta en la entrada de una compuerta OR se comporta como un 0. Por último, una salida V (verdadera) indica que la otra entrada de la compuerta XOR correspondiente se debe conectar a 0, y una salida C (complemento) especifica una conexión a 1.

El tamaño de un PLA se especifica dando el número de entradas, el número de términos producto y el número de salidas. Un circuito integrado PLA típico podría tener 16 entradas, 48 términos producto y 8 salidas. Para n entradas, k términos producto y m salidas, la lógica interna del PLA consiste en n compuertas búfer-inversor, k compuertas AND, m compuertas OR y m compuertas XOR. Hay $2n \times k$ conexiones entre las entradas y el arreglo AND, $k \times m$ conexiones entre los arreglos AND y OR, y m conexiones asociadas a las compuertas XOR.

Al diseñar un sistema digital con un PLA, no hay necesidad de indicar las conexiones internas de la unidad como hicimos en la figura 7-14. Lo único que se necesita es una tabla de programación de PLA con la cual el PLA se programará para que proporcione la lógica reque-

rida. Al igual que las ROM, el PLA puede ser programable por máscara o programable en el campo. Con programación por máscara, el cliente presenta al fabricante una tabla de programación de PLA, que el fabricante utiliza para producir un PLA a la medida que tiene la lógica interna especificada por el cliente. Un segundo tipo de PLA se llama arreglo de lógica programable en el campo (FPLA, *field programmable logic array*). El usuario programa el FPLA con una unidad programadora que se vende en el comercio.

Al implementar un circuito combinacional con un PLA, se debe efectuar una investigación cuidadosa para reducir el número de términos producto distintos, ya que el PLA tiene un número finito de compuertas AND. Esto se efectúa simplificando cada función booleana al número mínimo de términos. El número de literales en los términos no es importante porque ya se cuenta con todas las variables de entrada. Para cada función, se debe simplificar tanto la verdadera como su complemento, para ver cuál se puede expresar con menos términos producto y cuál genera términos productos comunes a otras funciones.

EJEMPLO 7-2

Implemente estas dos funciones booleanas con un PLA:

$$F_1(A, B, C) = \sum(0, 1, 2, 4)$$
$$F_2(A, B, C) = \sum(0, 5, 6, 7)$$

Las dos funciones se simplifican en los mapas de la figura 7-15. Se han simplificado en suma de productos tanto las funciones verdaderas como sus complementos. La combinación que da el mínimo de términos producto es

$$F_1 = (AB + AC + BC)'$$

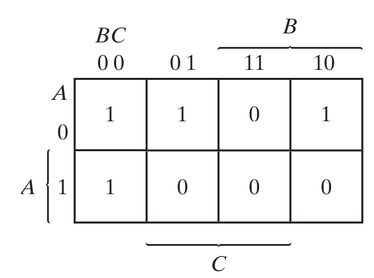
y

$$F_2 = AB + AC + A'B'C'$$

Esto da cuatro términos producto distintos: AB, AC, BC y $A'B'C'$. La tabla de programación de PLA se presenta en la figura. Cabe señalar que F_1 es la salida verdadera, aunque se ha marcado una C arriba de ella en la tabla. El motivo es que F_1 se genera con un circuito AND-OR y está disponible en la salida de la compuerta OR. La compuerta XOR complementa la función para producir la verdadera salida F_1.

■

El circuito combinacional empleado en el ejemplo 7-2 es demasiado simple como para implementarlo con un PLA; se presenta únicamente como ilustración. Un PLA típico tiene un gran número de entradas y términos producto. La simplificación de funciones booleanas con tantas variables necesita efectuarse con procedimientos de simplificación asistidos por computadora. El programa de diseño asistido por computadora simplifica cada función y su complemento al mínimo de términos. Luego, el programa selecciona el mínimo de términos producto que cubren todas las funciones en su forma verdadera o de complemento. Después se genera la tabla de programación y se obtiene el mapa de fusibles requerido. El mapa se aplica a un programador de FPLA que efectúa el procedimiento en hardware de quemar los fusibles internos del circuito integrado.



$$F_1 = A'B' + A'C' + B'C'$$
$$F_1 = (AB + AC + BC)'$$

A \ BC	00	01	11	10
0	1	0	0	0
1	0	1	1	1

$$F_2 = AB + AC + A'B'C'$$
$$F_2 = (A'C + A'B + AB'C')'$$

Tabla de programación de PLA

	Término producto	Entradas A	B	C	Salidas (C) F_1	(T) F_2
AB	1	1	1	–	1	1
AC	2	1	–	1	1	1
BC	3	–	1	1	1	–
$A'B'C'$	4	0	0	0	–	1

FIGURA 7-15
Solución del ejemplo 7-2

7-7 ARREGLO LÓGICO PROGRAMABLE

El arreglo lógico programable (PAL) es un dispositivo lógico programable con un arreglo OR fijo y un arreglo AND programable. Dado que sólo las compuertas AND son programables, el PAL es más fácil de programar, pero no es tan flexible como el PLA. La figura 7-16 ilustra la configuración lógica de un PAL representativo. Tiene cuatro entradas y cuatro salidas. Cada entrada tiene una compuerta búfer-inversor y cada salida se genera con una compuerta OR fija. La unidad tiene cuatro secciones, cada una de las cuales se compone de un arreglo AND-OR de anchura tres. Esta caracterización indica que hay tres compuertas AND programables en cada sección y una compuerta OR fija. Cada compuerta AND tiene 10 conexiones de entrada programables. Esto se indica en el diagrama con 10 líneas verticales que intersecan todas las líneas horizontales. La línea horizontal representa la configuración de múltiples entradas de la compuerta AND. Una de las salidas se conecta a una compuerta búfer-inversor y se realimenta a dos entradas de las compuertas AND.

Los dispositivos PAL comerciales contienen más compuertas que el que aparece en la figura 7-16. Un circuito integrado PAL típico podría tener ocho entradas, ocho salidas y ocho secciones, cada una con un arreglo AND-OR de anchura ocho. Las terminales de salida a veces se alimentan con búferes o inversores de tres estados.

Al diseñar con un PAL, las funciones booleanas deben simplificarse de modo que encajen en cada sección. A diferencia de los PLA, no es posible compartir términos producto entre dos

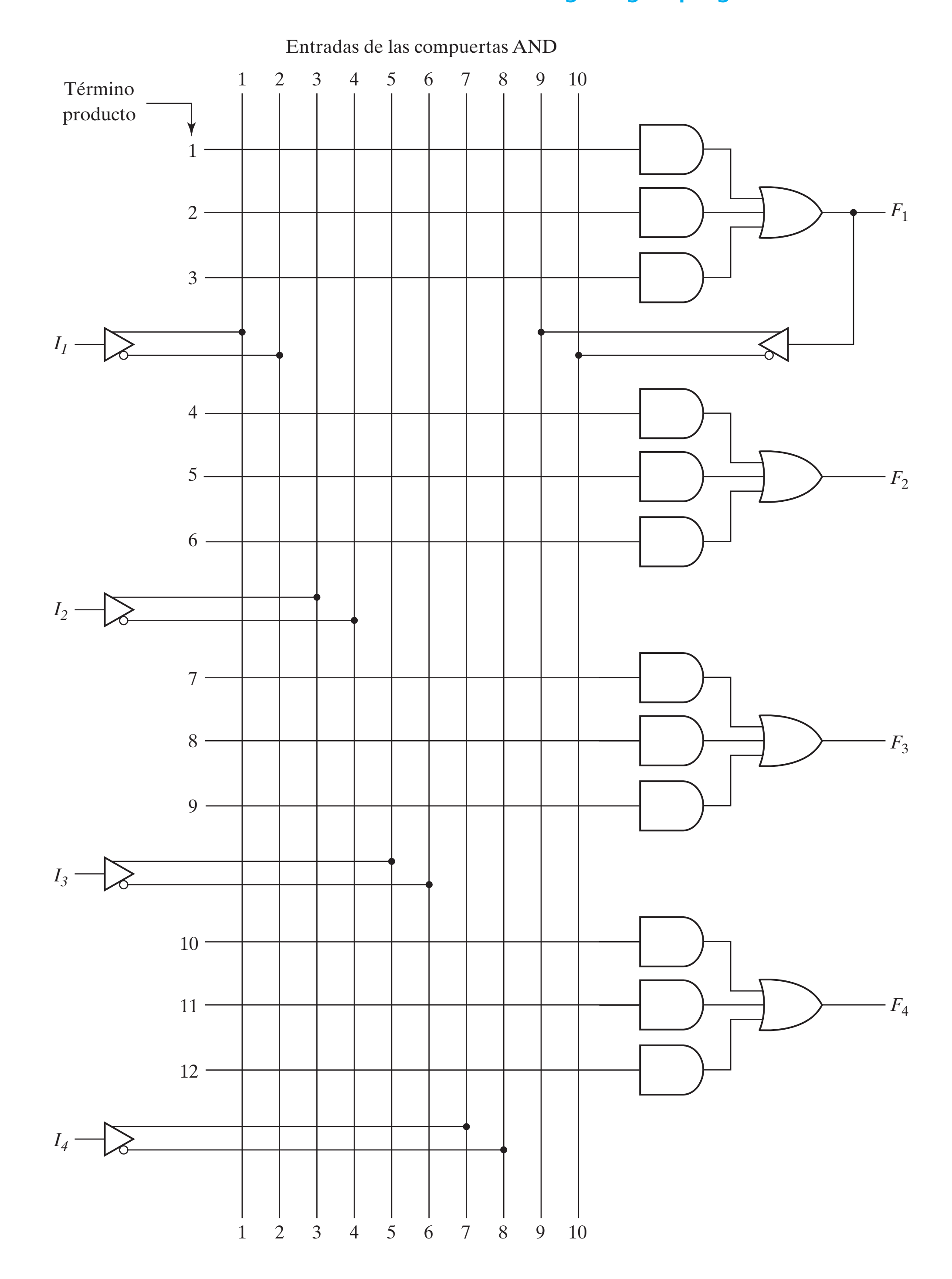


FIGURA 7-16
PAL con cuatro entradas, cuatro salidas y una estructura AND-OR de anchura tres

o más compuertas OR. Por ello, cada función se simplifica sola sin considerar los términos producto comunes. El número de términos producto en cada sección es fijo, y si el número de términos en la función es demasiado grande, podría ser necesario usar dos secciones para implementar una función booleana.

Como ejemplo del uso de PAL en el diseño de circuitos combinacionales, consideremos las siguientes funciones booleanas, dadas en suma de minitérminos:

$$w(A, B, C, D) = \sum(2, 12, 13)$$
$$x(A, B, C, D) = \sum(7, 8, 9, 10, 11, 12, 13, 14, 15)$$
$$y(A, B, C, D) = \sum(0, 2, 3, 4, 5, 6, 7, 8, 10, 11, 15)$$
$$z(A, B, C, D) = \sum(1, 2, 8, 12, 13)$$

La simplificación de las cuatro funciones al mínimo de términos produce estas funciones booleanas:

$$w = ABC' + A'B'CD'$$
$$x = A + BCD$$
$$y = A'B + CD + B'D'$$
$$z = ABC' + A'B'CD' + AC'D' + A'B'C'D$$
$$= w + AC'D' + A'B'C'D$$

Vemos que la función para z tiene cuatro términos producto. La suma lógica de dos de estos términos es igual a w. Si se usa w, es posible reducir el número de términos de z, de cuatro a tres.

La tabla de programación de PAL es similar a la que usamos para los PLA, con la salvedad de que sólo es necesario programar las entradas de las compuertas AND. La tabla 7-6 presenta la tabla de programación de PAL para las cuatro funciones booleanas. La tabla se divide en

Tabla 7-6
Tabla de programación de PAL

	Entradas de AND					
Término producto	***A***	***B***	***C***	***D***	***W***	**Salidas**
1	1	1	0	–	–	$w = ABC'$
2	0	0	1	0	–	$+ A'B'CD'$
3	–	–	–	–	–	
4	1	–	–	–	–	$x = A$
5	–	1	1	1	–	$+ BCD$
6	–	–	–	–	–	
7	0	1	–	–	–	$y = A'B$
8	–	–	1	1	–	$+ CD$
9	–	0	–	0	–	$+ B'D'$
10	–	–	–	–	1	$z = w$
11	1	–	0	0	–	$+ AC'D'$
12	0	0	0	1	–	$+ A'B'C'D$

cuatro secciones con tres términos producto en cada una, en correspondencia con el PAL de la figura 7-16. Las primeras dos secciones necesitan únicamente dos términos producto para implementar la función booleana. La última sección, la de la salida z, necesita cuatro términos producto. Si utilizamos la salida de w, podremos reducir la función a tres términos.

El mapa de fusibles del PAL especificado por la tabla de programación se observa en la figura 7-17. Por cada 1 o 0 en la tabla, marcamos la intersección correspondiente en el diagrama con el símbolo de fusible intacto. Por cada guión, marcamos el diagrama con fusibles quemados en ambas entradas, verdadera y complemento. Si una compuerta AND no se usa, dejamos intactos todos sus fusibles de entrada. Puesto que la entrada correspondiente recibe tanto la variable de entrada verdadera como su complemento, tendremos $AA' = 0$ y la salida de la compuerta AND siempre será 0.

Al igual que con todos los PLD, el diseño con PAL se facilita si se emplean técnicas de diseño asistidas por computadora. El quemado de fusibles internos es un procedimiento en hardware que se efectúa con la ayuda de instrumentos electrónicos especiales.

7-8 DISPOSITIVOS PROGRAMABLES SECUENCIALES

Los sistemas digitales se diseñan empleando flip-flops y compuertas. Puesto que el PLD combinacional sólo contiene compuertas, es necesario incluir flip-flops externos cuando se utilizan en el diseño. Los dispositivos programables secuenciales contienen tanto compuertas como flip-flops. Así, el dispositivo puede programarse para que realice diversas funciones de circuito secuencial. En el comercio hay varios tipos de dispositivos programables secuenciales, y cada uno tiene variantes dentro de cada tipo que son específicas para el proveedor dado. La lógica interna de estos dispositivos es demasiado compleja como para mostrarse aquí; por ello, describiremos tres tipos principales sin entrar en los pormenores de su construcción:

1. Dispositivo lógico programable secuencial (o simple) (SPLD)
2. Dispositivo lógico programable complejo (CPLD)
3. Arreglo de compuertas programable en el campo (FPGA)

El PLD secuencial también se describe como "simple" para distinguirlo del PLD complejo. Los SPLD incluyen flip-flops dentro del chip de circuitos integrados además del arreglo AND-OR. El resultado es un circuito secuencial como el que se aprecia en la figura 7-18. Un PAL o PLA se modifica por la inclusión de varios flip-flops conectados para formar un registro. Las salidas del circuito se toman de las compuertas OR o de las salidas de los flip-flops. Hay conexiones programables adicionales que permiten incluir las salidas de los flip-flops en los términos producto que se forman con el arreglo AND. Los flip-flops pueden ser del tipo D o del tipo JK.

El primer dispositivo programable que se creó para apoyar la implementación de circuitos secuenciales fue el secuenciador lógico programable en el campo (FPLS, *field-programmable logic sequence*). Un FPLS típico se organiza en torno a un PLA con varias salidas que alimentan a flip-flops. Éstos son flexibles en cuanto a que es posible programarlos de modo que operen como de tipo JK o de tipo D. El FPLS no tuvo éxito comercial porque tiene demasiadas conexiones programables. La configuración que se emplea generalmente para los SPLD es el PAL combinacional junto con flip-flops D. Un PAL que incluye flip-flops se conoce como PAL con registros para indicar que el dispositivo contiene flip-flops además del arreglo AND-OR. Cada sección de un SPLD se denomina *macrocelda*. Una macrocelda es un circuito que contiene una función de lógica combinacional de suma de productos y un flip-flop opcional. Supondremos una suma de productos AND-OR, pero en la práctica puede ser cualquiera de las implementaciones de dos niveles que se presentaron en la sección 3-7.

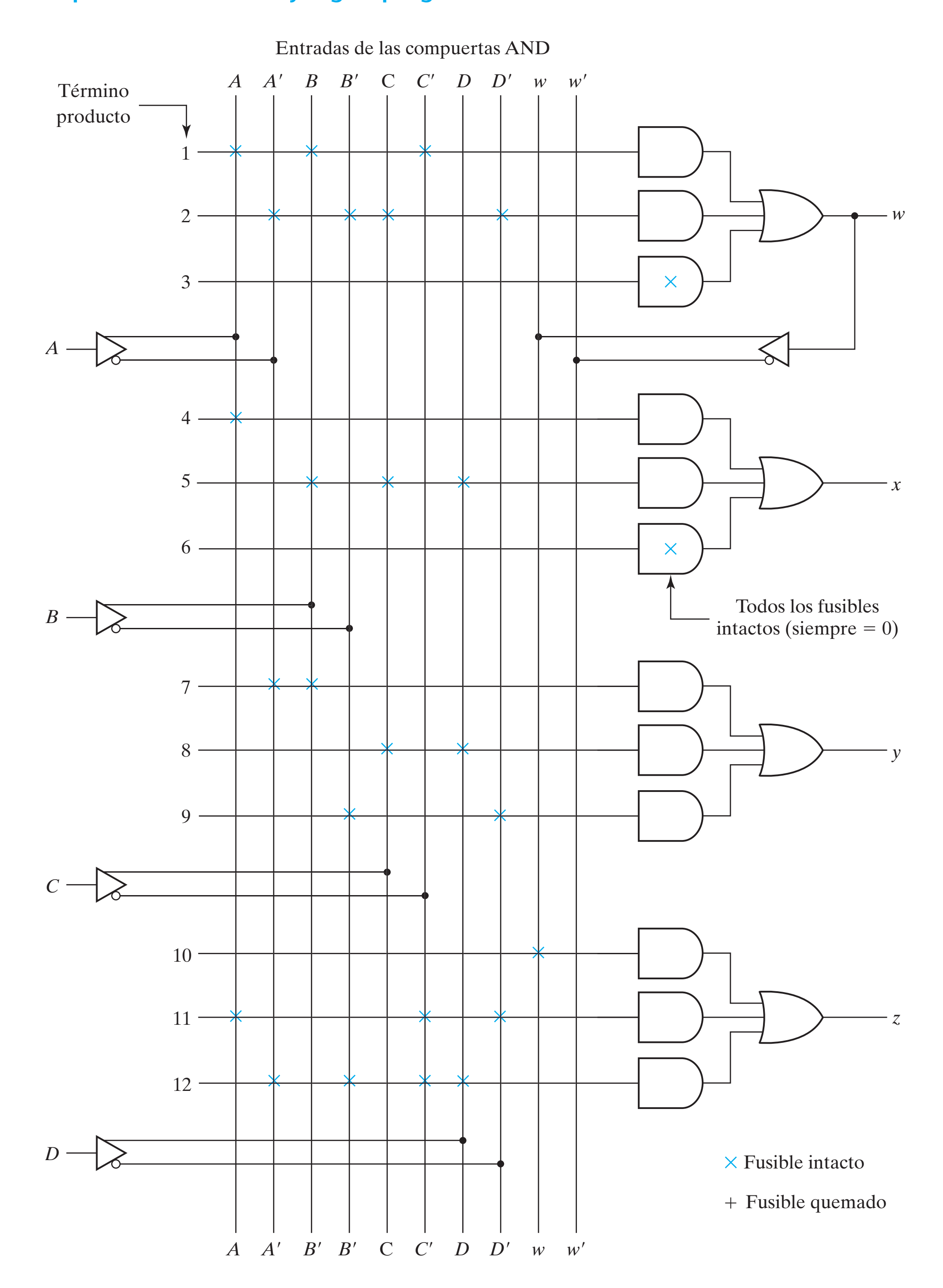


FIGURA 7-17
Mapa de fusibles del PAL especificado en la tabla 7-6

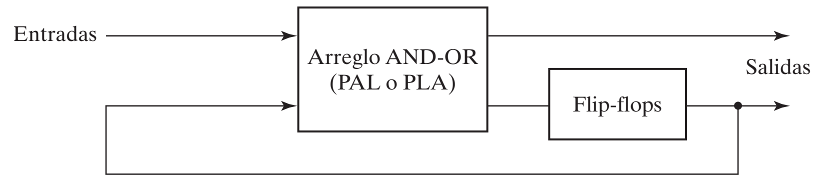


FIGURA 7-18
Dispositivo lógico programable secuencial

La figura 7-19 muestra la lógica de una macrocelda básica. El arreglo AND-OR es el mismo que el del PAL combinacional que se observa en la figura 7-16. La salida se alimenta con un flip-flop *D* disparado por flanco. El flip-flop está conectado a una entrada de reloj común y cambia de estado en un borde de reloj. La salida del flip-flop se conecta a un búfer (o inversor) de tres estados controlado por una señal de habilitación de salida (*output enable*) marcada en el diagrama con OE. La salida del flip-flop se realimenta a una de las entradas de las compuertas AND programables para suministrar la condición de estado actual del circuito secuencial. Un SPLD típico tiene entre ocho y diez macroceldas en un paquete de CI. Todos los flip-flops están conectados a la entrada de reloj CLK común, y todos los búferes de tres estados se controlan con la entrada OE.

Además del arreglo AND programable, la macrocelda podría tener otras características programables. Entre las opciones de programación más comunes están la capacidad de usar el flip-flop o bien pasarlo por alto, seleccionar la polaridad del borde de reloj, seleccionar preestablecimiento (*preset*) y despeje (*clear*) para el registro, y seleccionar la forma verdadera o el complemento de una salida. Se utiliza una compuerta XOR para programar una condición de

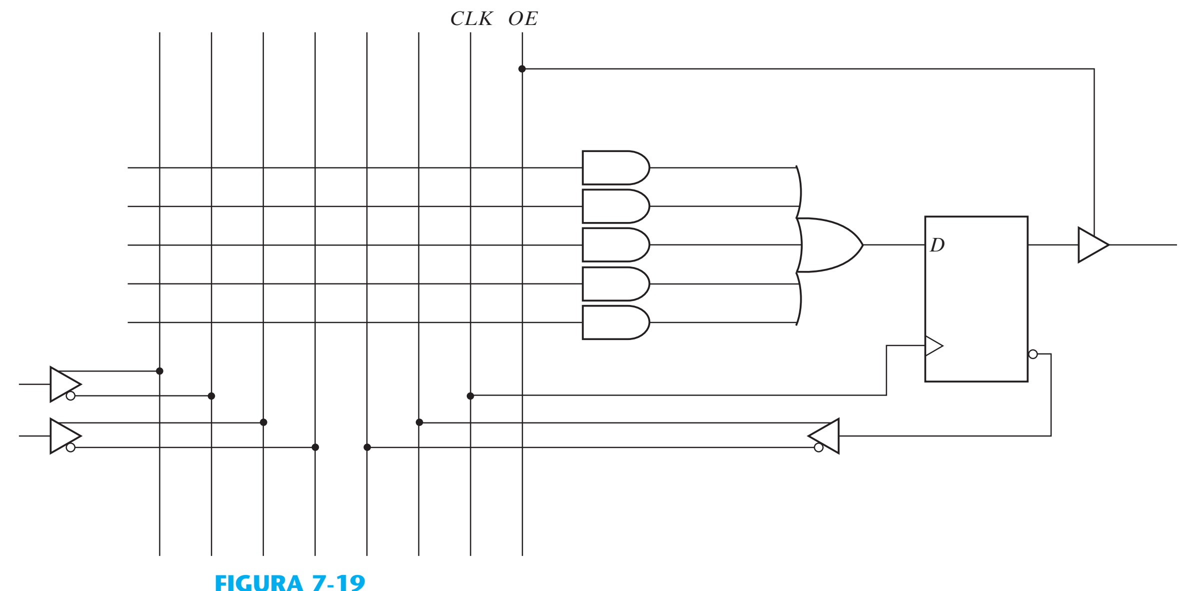


FIGURA 7-19
Lógica de una macrocelda básica

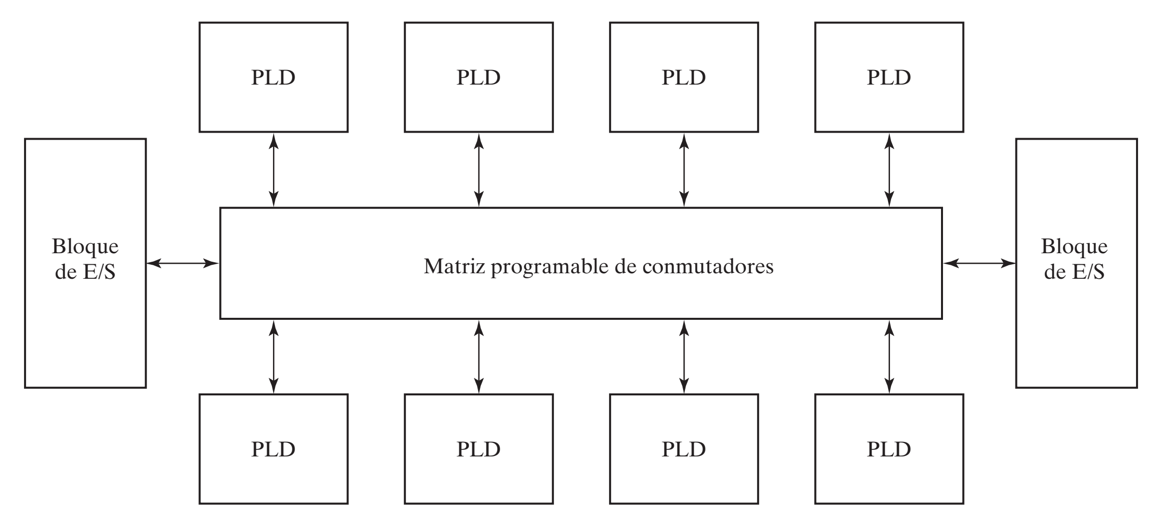


FIGURA 7-20
Configuración general de un CPLD

verdadera/complemento. Los multiplexores permiten escoger entre dos o cuatro trayectorias distintas programando las entradas de selección.

El diseño de un sistema digital con PLD suele requerir la conexión de varios dispositivos para producir la especificación completa. En este tipo de aplicaciones, resulta más económico utilizar un dispositivo lógico programable complejo (CPLD). Un CPLD es un conjunto de PLD individuales en un solo circuito integrado. Una estructura de interconexión programable permite conectar los PLD entre sí de la misma manera que se hace con PLD individuales.

La figura 7-20 muestra la configuración general de un CPLD. Consta de múltiples PLD interconectados a través de una matriz programable de conmutadores. Los bloques de entrada/salida (E/S) proporcionan las conexiones a las terminales del CI. Cada terminal de E/S se alimenta con un búfer de tres estados y se puede programar de modo que actúe como entrada o como salida. La matriz de conmutadores recibe entradas del bloque de E/S y las dirige a las macroceldas individuales. De forma similar, salidas selectas de las macroceldas se envían a las salidas según sea necesario. Cada PLD suele contener entre 8 y 16 macroceldas. Las macroceldas dentro de cada PLD por lo regular están plenamente interconectadas. Si una macrocelda tiene términos producto no utilizados, otras macroceldas cercanas pueden usarlos. En algunos casos, el flip-flop de la macrocelda se programa para que actúe como flip-flop *D*, *JK* o *T*.

Los diferentes fabricantes han adoptado distintos enfoques respecto a la arquitectura general de los CPLD. Entre los aspectos en los que difieren están los PLD individuales (también llamados *bloques de función*), el tipo de macroceldas, los bloques de E/S y la estructura programable de interconexión. La mejor forma de investigar un dispositivo específico de un proveedor es estudiar la literatura del fabricante.

El componente básico empleado en un diseño VLSI es el *arreglo de compuertas*, que consiste en un patrón de compuertas fabricadas en un área de silicio y que se repite miles de veces hasta cubrir todo el chip con las compuertas. Se fabrican arreglos de entre mil y cien mil compuertas en un solo chip de CI, dependiendo de la tecnología empleada. El diseño con arreglos de compuertas requiere que el cliente proporcione al fabricante el patrón de interco-

nexión deseado. Los primeros niveles del proceso de fabricación son comunes e independientes de la función lógica final. Se requieren pasos de fabricación adicionales para interconectar las compuertas según las especificaciones del diseñador.

Un arreglo de compuertas programable en el campo (FPGA, *field programmable gate array*) es un circuito VLSI que se programa en las instalaciones del cliente. Un FPGA típico consiste en un arreglo de cientos o miles de bloques lógicos, rodeado por bloques programables de entrada/salida y conectado mediante interconexiones programables. Existe una amplia variedad de configuraciones internas dentro de este grupo de dispositivos. El desempeño de cada tipo de dispositivos depende del circuito contenido en sus bloques lógicos y de la eficiencia de sus interconexiones programadas.

Un bloque lógico de FPGA por lo regular consiste en tablas de consulta, multiplexores, compuertas y flip-flops. La tabla de consulta es una tabla de verdad almacenada en una SRAM, y proporciona las funciones de circuito combinacional del bloque lógico. Estas funciones se implementan a partir de la tabla de verdad almacenada en la SRAM, de manera similar a la forma en que se implementan funciones de circuito combinacional con ROM, que se describió en la sección 7.5. Por ejemplo, una SRAM de 16×2 permite almacenar la tabla de verdad de un circuito combinacional que tiene cuatro entradas y dos salidas. Se utiliza la sección de lógica combinacional, junto con varios multiplexores programables, para configurar las ecuaciones de entrada del flip-flop y la salida del bloque lógico.

La ventaja de usar RAM en lugar de ROM para almacenar la tabla de verdad es que la tabla puede programarse escribiendo en la memoria. La desventaja es que la memoria es volátil y hay que volver a cargar el contenido de la tabla de consulta si llega a interrumpirse la alimentación eléctrica. El programa se baja de una computadora anfitriona o de una PROM en la misma tarjeta. El programa permanece en SRAM hasta que el FPGA se reprograma o se apaga el equipo. El dispositivo debe reprogramarse cada vez que se enciende el equipo. La capacidad de reprogramar el FPGA es útil en diversas aplicaciones en las que se requieren diferentes implementaciones lógicas en el programa.

El diseño con PLD, CPLD o FPGA requiere muchas herramientas de diseño asistido por computadora (CAD, *computer-aided design*) que faciliten el procedimiento de síntesis. Se cuenta con diversas herramientas, como paquetes para captura de diagramas y lenguajes para describir hardware (HDL, *hardware description languages*) como ABEL, VHDL y Verilog. Existen herramientas de síntesis que asignan, configuran y conectan bloques lógicos para hacerlos corresponder con una descripción de diseño de alto nivel escrita en HDL.

PROBLEMAS

7-1 Las siguientes unidades de memoria se especifican por el número de palabras multiplicado por el número de bits por palabra. ¿Cuántas líneas de dirección y líneas de entrada-salida de datos se necesitan en cada caso? a) $4K \times 16$, b) $2G \times 8$, c) $16M \times 32$, d) $256K \times 64$.

7-2 Indique el número de bytes que se almacenan en las memorias del problema 7-1.

7-3 La palabra número 723 de la memoria que se muestra en la figura 7-3 contiene el equivalente binario de 3451. Dé la dirección de 10 bits y los 16 bits del contenido de esa palabra de memoria.

7-4 Muestre las formas de onda de temporización de ciclos de memoria para las operaciones de escritura y lectura. Suponga un reloj de CPU de 25 MHz y un tiempo de ciclo de memoria de 60 ns.

7-5 Escriba un conjunto de pruebas para la memoria que se describe en el ejemplo HDL 7-1. El programa de prueba almacena un 7 binario en la dirección 0 y un 14 binario en la dirección 60. Luego se leen esas dos direcciones para verificar su contenido.

7-6 Encierre la RAM de 4 × 4 de la figura 7-6 en un diagrama de bloques que señale todas las entradas y salidas. Suponiendo salidas de tres estados, construya una memoria de 8 × 8 utilizando cuatro unidades de RAM de 4 × 4.

7-7 Una memoria de 16K × 4 utiliza decodificación coincidente dividiendo el decodificador interno en selección X y selección Y.

a) ¿Qué tamaño tiene cada decodificador y cuántas compuertas AND se requieren para decodificar la dirección?

b) Determine las líneas de selección X y Y que están habilitadas cuando la dirección de entrada es el equivalente binario de 6000.

7-8 a) ¿Cuántos chips de RAM de 32K × 8 se necesitan para tener una capacidad de memoria de 256K bytes?

b) ¿Cuántas líneas de dirección se necesitan para acceder a 256K bytes? ¿Cuántas de esas líneas están conectadas a las entradas de dirección de todos los chips?

c) ¿Cuántas líneas deben decodificarse para las entradas de selección de chip? Especifique el tamaño del decodificador.

7-9 Un chip de DRAM utiliza multiplexión bidimensional de direcciones. Tiene 13 terminales de dirección comunes, y la dirección de fila tiene un bit más que la dirección de columna. ¿Qué capacidad tiene la memoria?

7-10 Dada la palabra de datos 01011011, de ocho bits, genere la palabra compuesta de 13 bits correspondiente al código Hamming que corrige errores individuales y detecta errores dobles.

7-11 Deduzca la palabra de código Hamming de 15 bits para la palabra de datos de 11 bits 11001001010.

7-12 Se lee de la memoria una palabra de código Hamming de 12 bits que contiene ocho bits de datos y cuatro bits de paridad. Indique la palabra de datos original, de ocho bits, que se escribió en la memoria, si la palabra de 12 bits que se lee es:

a) 000011101010   b) 101110000110

c) 101111110100

7-13 ¿Cuántos bits de comprobación de paridad deben incluirse junto con la palabra de datos para poder corregir errores individuales y detectar errores dobles si la palabra de datos contiene a) 16 bits, b) 32 bits y c) 48 bits.

7-14 Es necesario formular el código Hamming para cuatro bits de datos, D_3, D_5, D_6 y D_7, junto con tres bits de paridad, P_1, P_2 y P_4.

a) Evalúe la palabra compuesta de siete bits para la palabra de datos 0010.

b) Evalúe tres bits de comprobación, C_4, C_2 y C_1, suponiendo que no hay error.

c) Suponga que se produce un error en el bit D_5 al escribir la palabra en la memoria. Explique cómo se detecta y corrige el error de ese bit.

d) Añada el bit de paridad P_8 para incluir en el código detección de errores dobles. Suponga que hubo errores en los bits P_2 y D_5. Explique cómo se detecta el doble error.

7-15 Dado un chip de ROM de 32 × 8 con una entrada de habilitación, indique las conexiones externas que se requieren para construir una ROM de 128 × 8 con cuatro chips y un decodificador.

7-16 Un chip de ROM de 4096 × 8 bits tiene dos entradas de selección de chip y opera con una fuente de poder de 5 volts. ¿Cuántas terminales necesita el paquete de circuito integrado? Dibuje un diagrama de bloques y rotule todas las terminales de entrada y salida de la ROM.

7-17 Una ROM de 32 × 6 junto con una línea 2^0 (véase la figura P7-17) convierten un número binario de seis bits en el número BCD de dos dígitos correspondiente. Por ejemplo, el número binario 100001 se convierte en el número BCD 011 0011 (decimal 33). Especifique la tabla de verdad de la ROM.

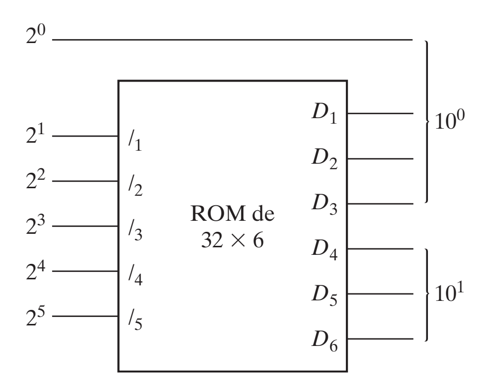


FIGURA P7-17

7-18 Especifique el tamaño de una ROM (número de palabras y número de bits por palabra) que dará cabida a la tabla de verdad de los componentes de circuito combinacional que se indican a continuación:

a) un multiplicador binario que multiplica dos números de cuatro bits,

b) un sumador-restador de cuatro bits,

c) multiplexores cuádruples de 2 líneas a 1 con entradas de selección y habilitación comunes, y

d) un decodificador de BCD a siete segmentos con una entrada de habilitación.

7-19 Prepare la tabla de verdad de una ROM de 8 × 4 que implementa las funciones booleanas

$$A(x, y, z) = \Sigma(1, 2, 4, 6)$$
$$B(x, y, z) = \Sigma(0, 1, 6, 7)$$
$$C(x, y, z) = \Sigma(2, 6)$$
$$D(x, y, z) = \Sigma(1, 2, 3, 5, 7)$$

Ahora considere la ROM como memoria y especifique el contenido de las direcciones de memoria 1 y 4.

7-20 Prepare la tabla de programación de PLA para las cuatro funciones booleanas del problema 7-19. Reduzca al mínimo el número de términos producto.

7-21 Deduzca la tabla de programación de PLA para el circuito combinacional que eleva al cuadrado un número de tres bits. Reduzca al mínimo el número de términos producto. (La implementación equivalente en ROM se da en la figura 7-12.)

7-22 Prepare la tabla de programación de PLA del convertidor de BCD a código exceso 3 cuyas funciones booleanas se simplifican en la figura 4-3.

7-23 Repita el problema 7-22 empleando un PAL.

7-24 La que sigue es la tabla de verdad de un circuito combinacional con tres entradas y cuatro salidas. Prepare la tabla de programación de PAL para el circuito y marque el mapa de fusibles en un diagrama de PAL similar al de la figura 7-17.

Entradas			Salidas			
x	**y**	**z**	**A**	**B**	**C**	**D**
0	0	0	0	1	0	0
0	0	1	1	1	1	1
0	1	0	1	0	1	1
0	1	1	0	1	0	1
1	0	0	1	0	1	0
1	0	1	0	0	0	1
1	1	0	1	1	1	0
1	1	1	0	1	1	1

7-25 Utilizando la macrocelda con registro de la figura 7-19, muestre el mapa de fusibles de un circuito secuencial con dos entradas, x y y, y un flip-flop, A, descrito por la ecuación de entrada

$$D_A = x \oplus y \oplus A$$

7-26 Modifique el diagrama de PAL de la figura 7-16 incluyendo tres flip-flops tipo D con reloj, entre las compuertas OR y las salidas, como en la figura 7-19. El diagrama deberá ajustarse al diagrama de bloques de un circuito secuencial. Ello requerirá compuertas búfer-inversor adicionales y seis líneas verticales para conectar las salidas de flip-flop con el arreglo AND a través de conexiones programables. Utilizando el diagrama de PAL con registro modificado, muestre el mapa de fusibles que implementa un contador binario de tres bits con acarreo de salida.

REFERENCIAS

1. Tocci R. J. y N. S. Widmer. 2001. *Digital Systems Principles and Applications*, 8a. ed. Upper Saddle River, NJ: Prentice-Hall.
2. Kitson, B. 1984. *Programmable Array Logic Handbook*. Sunnyvale, CA: Advanced Micro Devices.
3. Wakerly, J. F. 2000. *Digital Design: Principles and Practices*. 3a. ed. Upper Saddle River, NJ: Prentice-Hall.
4. Nelson, V. P., H. T. Nagle, J. D. Irwin y B. D. Carroll 1997. *Digital Logic Circuit Analysis and Design*. Upper Saddle River, NJ: Prentice-Hall.
5. Hamming, R. W. 1950. Error Detecting and Error Correcting Codes. *Bell Syst. Tech. J.* 29: 147-160.
6. Lin, S. y D. J. Costello, Jr. 1983. *Error Control Coding*. Englewood Cliffs, NJ: Prentice-Hall.
7. 1988. *Programmable Logic Data Book*. Dallas: Texas Instruments.
8. Trimberger, S. M. 1994. *Field Programmable Gate Array Technology*. Boston: Kluwer Academic Pub.
9. 1994. *The Programmable Logic Data Book*, 2a. ed. San José, CA: Xilinx, Inc.
10. 1986. *Memory Components Handbook*. Santa Clara, CA: Intel.

8 Nivel de transferencia de registros

8-1 NOTACIÓN DE NIVEL DE TRANSFERENCIA DE REGISTROS (RTL)

Un sistema digital es un sistema de lógica secuencial construido con flip-flops y compuertas. Los circuitos secuenciales se pueden especificar con tablas de estados como se explicó en el capítulo 5. Especificar un sistema digital grande con una tabla de estados es muy difícil, si no imposible, porque el número de estados sería prohibitivamente grande. Para superar esta dificultad, los sistemas digitales se diseñan con un enfoque modular. El sistema se divide en subsistemas modulares, cada uno de los cuales desempeña alguna tarea funcional. Los módulos se construyen de dispositivos digitales como registros, decodificadores, multiplexores, elementos de artimética y lógica de control. Los diversos módulos se interconectan con trayectorias de datos y de control comunes para formar un sistema digital.

La mejor forma de describir los módulos digitales es con un conjunto de registros y las operaciones que se efectúan con la información binaria almacenada en ellos. Como ejemplos de operaciones de registros podemos citar desplazamiento, conteo, despeje y carga. Se supone que los registros son los componentes básicos del sistema digital. El flujo de información y el procesamiento efectuado con los datos almacenados en los registros reciben el nombre de operaciones de transferencia de registros. Un sistema digital se representa en el *nivel de transferencia de registros* (RTL, *register transfer level*) cuando se especifica con estos tres componentes:

1. El conjunto de registros del sistema.
2. Las operaciones que se efectúan con los datos almacenados en los registros.
3. El control que supervisa la sucesión de operaciones del sistema.

Un registro es un grupo de flip-flops que almacena información binaria y puede realizar una o más operaciones elementales. Un registro es capaz de cargar nueva información o desplazar la información a la derecha o a la izquierda. Un contador es un registro que incrementa un número en uno. Un flip-flop solo se considera un registro de un bit que puede establecerse

(encenderse, ponerse en 1), despejarse (apagarse, ponerse en 0) o complementarse. De hecho, los flip-flops y las compuertas de cualquier circuito secuencial se pueden llamar registros según esta definición.

Las operaciones que se ejecutan con la información almacenada en registros son operaciones elementales que se efectúan en paralelo con una cadena de bits durante un ciclo de reloj. El resultado de la operación podría reemplazar la información binaria previa de un registro. Otra posibilidad es que el resultado se transfiera a otro registro, sin alterar los datos previos. Los circuitos digitales que presentamos en el capítulo 6 son registros que implementan operaciones elementales. Un contador con carga paralela puede efectuar las operaciones de incremento y carga. Un registro de desplazamiento bidireccional puede realizar las operaciones de desplazamiento a la derecha y desplazamiento a la izquierda.

El control que inicia la sucesión de operaciones consiste en señales de temporización que ordenan las operaciones de cierta manera. Ciertas condiciones que dependen de resultados de operaciones anteriores podrían determinar el orden de operaciones futuras. Las salidas de la lógica de control son variables binarias que inician las diversas operaciones en el registro.

La transferencia de información de un registro a otro se indica de forma simbólica con un operador de sustitución. El enunciado

$$R2 \leftarrow R1$$

denota la transferencia del contenido del registro $R1$ al registro $R2$. Especifica una sustitución del contenido de $R2$ por el contenido de $R1$. Por definición, el contenido del registro de origen $R1$ no cambia después de la transferencia. La flecha representa la transferencia y la dirección en que se efectúa.

Un enunciado que especifica una transferencia de registro implica que existen conexiones de circuito que van de las salidas del registro de origen a las entradas del registro de destino, y que el registro de destino tiene capacidad de carga en paralelo. Normalmente, no queremos que se efectúe la transferencia en cada ciclo de reloj, sino sólo cuando se da cierta condición predeterminada. Las condiciones se expresan con enunciados condicionales if-then (si-entonces), como

$$\text{If } (T1 = 1) \text{ then } (R2 \leftarrow R1)$$

donde $T1$ es una señal de control generada en la sección de control. Advierta que el reloj no se incluye como variable en los enunciados de transferencia de registro. Se supone que todas las transferencias se efectúan durante una transición de borde de reloj. Aunque es posible que una condición de control como $T1$ se cumpla antes de una transición de reloj, la transferencia no se efectuará sino hasta que haya terminado la transición de reloj.

Se puede usar una coma para separar dos o más operaciones que se ejecutan al mismo tiempo. Consideremos el enunciado

$$\text{If } (T3 = 1) \text{ then } (R2 \leftarrow R1, R1 \leftarrow R2)$$

Este enunciado denota una operación que intercambia el contenido de dos registros durante el mismo borde de reloj, a condición de que $T3 = 1$. Esta operación simultánea es posible con registros que tienen flip-flops disparados por flanco (borde). He aquí otros ejemplos de transferencias de registro:

$R1 \leftarrow R1 + R2$	Sumar el contenido de $R2$ a $R1$
$R3 \leftarrow R3 + 1$	Incrementar $R3$ en 1 (contar hacia arriba)
$R4 \leftarrow \text{shr } R4$	Desplazar $R4$ a la derecha
$R5 \leftarrow 0$	Despejar $R5$ (ponerlo en ceros)

La suma se efectúa con un sumador binario paralelo; el incremento, con un contador; y el desplazamiento con un registro de desplazamiento. Los tipos de operaciones que más comúnmente se efectúan en los sistemas digitales pueden pertenecer a una de cuatro categorías:

1. Operaciones de transferencia que transfieren datos de un registro a otro.
2. Operaciones aritméticas que efectúan cálculos simples con los datos de los registros.
3. Operaciones de lógica que manipulan los bits de datos no numéricos en los registros.
4. Operaciones de desplazamiento que desplazan los datos en los registros.

La operación de transferencia no altera la información que se traslada del registro de origen al de destino. Los otros tres tipos modifican la información durante la transferencia. La notación de transferencia de registros y los símbolos empleados para representar las diversas operaciones de transferencia de registros no están estandarizados. Aquí usaremos dos tipos de notaciones. La notación que presentamos en esta sección se usará informalmente para especificar y explicar sistemas digitales en el nivel de transferencia de registros. En la sección que sigue presentaremos los símbolos de RTL que se usan en Verilog HDL.

8-2 NIVEL DE TRANSFERENCIA DE REGISTROS EN HDL

Los sistemas digitales se pueden describenir en el nivel de transferencia de registros con un lenguaje de descripción de hardware. En Verilog HDL, las descripciones RTL utilizan una combinación de construcciones de comportamiento y de flujo de datos. Las transferencias de registro se especifican con enunciados de asignación procedimental. Las funciones de circuitos combinacionales se especifican con enunciados de asignación continua o procedimental. El símbolo empleado para designar una transferencia es el signo de igual o una flecha. La sincronización con el reloj se logra utilizando un enunciado **always** cuyo control de sucesos es **posedge** o **negedge.** Los ejemplos que siguen ilustran las formas en que se puede especificar una transferencia en Verilog HDL:

```
assign  S = A + B;          Asignación continua

always @ (A or B)           Asignación procedimental (sin reloj)
    S = A + B;

always @ (posedge clock)    Asignación procedimental bloqueadora
  begin
   RA = RA + RB;
   RD = RA;
  end

always @ (negedge clock)    Asignación procedimental no bloqueadora
  begin
   RA <= RA + RB;
   RD <= RA;
  end
```

Se utilizan asignaciones continuas para especificar circuitos combinacionales. El enunciado **assign** anterior describe un sumador binario con entradas *A* y *B* y salida *S*. El operando de

destino en un enunciado de asignación (*S* en este caso) no puede ser un registro. Las salidas de circuitos combinacionales se pueden transferir a un registro con una asignación procedimental con reloj. La asignación procedimental sin reloj del segundo ejemplo ilustra otra forma de especificar un circuito combinacional.

Hay dos clases de asignaciones procedimentales: *bloqueadoras* y *no bloqueadoras*. Las distinguimos por los símbolos que usan. Las asignaciones bloqueadoras usan el símbolo (=) como operador de transferencia; y las no bloqueadoras, el símbolo (<=). Los enunciados de asignación bloqueadores se ejecutan sucesivamente en el orden en que aparecen en el bloque secuencial. Los enunciados no bloqueadores evalúan las expresiones del miembro derecho, pero no efectúan la asignación al miembro izquierdo sino hasta que se han evaluado todas las expresiones. Consideremos los dos ejemplos anteriores. En la asignación procedimental bloqueadora, el primer enunciado transfiere la suma a *RA*, mientras que el segundo transfiere el nuevo valor de *RA* a *RD*. Al final, *RA* y *RD* tienen el mismo valor. En la asignación procedimental no bloqueadora, las dos operaciones se efectúan de forma concurrente, de modo que *RD* recibe el valor original de *RA*.

Para garantizar la sincronía de las operaciones en un diseño RTL, es preciso usar asignaciones procedimentales no bloqueadoras para las variables que siguen a un enunciado always con reloj. El objetivo es evitar cualquier posibilidad de incongruencia funcional entre el modelo de diseño y la descripción HDL. La asignación no bloqueadora que aparece en un enunciado **always** con reloj modela correctamente el comportamiento de un circuito secuencial sincrónico.

Operadores de HDL

En la tabla 8-1 se presentan los operadores de Verilog HDL que se emplean en el diseño RTL, junto con sus símbolos. Los operadores de aritmética, lógica y desplazamiento son necesarios para describir operaciones de transferencia de registros. Los operadores lógicos y relacionales son útiles para especificar condiciones de control. Las operacioness aritméticas se efectúan con números binarios. Los números negativos se representan en complemento a dos. El operador de residuo produce el residuo de la división de dos números. Por ejemplo, 14 % 3 da 2.

Hay dos tipos de operadores de lógica: bit por bit y de reducción. Los operadores bit por bit (*bit-wise*) efectúan una operación lógica sucesivamente con los bits de dos operandos. Toman cada uno de los bits de un operando y efectúan la operación con el bit correspondiente del otro operando. Los operadores de reducción efectúan la operación lógica con un solo operando. Efectúan la operación bit por bit de derecha a izquierda y producen un resultado de un solo bit. Por ejemplo, la reducción NOR (~ |) produce 0 con el operando 00101 y 1 con el operando 00000. La negación no se usa como operador de reducción. En la tabla 4-9 de la sección 4-11 se presentan las tablas de verdad de los operadores bit por bit.

Los operadores lógicos y relacionales pueden tomar variables o expresiones como operandos. Básicamente, sirven para determinar condiciones verdaderas o falsas. Su evaluación da 1 si la condición es verdadera, y 0 si es falsa. Si la condición es ambigua, el resultado de su evaluación es x. Si el operando es un número, la evaluación da 0 si el número es igual a cero, y 1 si el número es distinto de cero. Por ejemplo, si $A = 1010$ y $B = 0000$, *A* se toma como 1 (el número es distinto de cero) y *B* se toma como 0. Los resultados de otras operaciones con estos valores son:

```
A && B = 0
A || B = 1
!A = 0
!B = 1
(A > B) = 1
(A == B)= 0
```

Tabla 8-1
Operadores de Verilog HDL

Tipo de operador	Símbolo	Operación efectuada
Aritmética	+	suma
	−	resta
	*	multiplicación
	/	división
	%	residuo
Lógica	~	negación (complemento)
(bit por bit	&	AND
o	|	OR
reducción)	^	OR exclusivo (XOR)
Lógicos	!	negación
	&&	AND
	||	OR
De desplazamiento	>>	desplazamiento a la derecha
	<<	desplazamiento a la izquierda
	{ , }	concatenación
Relacionales	>	mayor que
	<	menor que
	==	igualdad
	!=	desigualdad
	>=	mayor o igual que
	<=	menor o igual que

Los operadores de desplazamiento desplazan un operando vector a la derecha o a la izquierda cierto número de bits. Las posiciones de bit que quedan vacantes se llenan con ceros. Por ejemplo, si $R = 11010$, el enunciado

```
R = R >> 1;
```

desplaza R una posición a la derecha. El nuevo valor de R es 01101. El operador de concatenación permite juntar varios operandos. Puede servir para especificar un desplazamiento incluyendo los bits que se transfieren en las posiciones vacantes, como se vio. Ya vimos esto en el ejemplo HDL 6-1 para el registro de desplazamiento.

Enunciados cíclicos

Verilog HDL tiene cuatro tipos de ciclos que permiten ejecutar repetidamente enunciados procedimentales: *repeat*, *forever*, *while* y *for*. Todos los enunciados cíclicos deben aparecer dentro de un bloque **initial** o **always.**

El ciclo **repeat** ejecuta los enunciados correspondientes cierto número de veces. Ya usamos antes el ejemplo siguiente:

```
initial
   begin
       clock = 1'b0;
       repeat (16)
       #5 clock = ~ clock;
   end
```

Esto produce ocho ciclos de reloj con un tiempo de ciclo de 10 unidades de tiempo.

El ciclo **forever** hace que se ejecute continuamente el enunciado procedimental una y otra vez. Por ejemplo, el ciclo siguiente produce un reloj continuo:

```
initial
   begin
     clock = 1'b0;
     forever
     # clock = ~ clock;
   end
```

El ciclo **while** ejecuta un enunciado o bloque de enunciados una y otra vez en tanto una expresión sea verdadera. Si la expresión es falsa inicialmente, el enunciado nunca se ejecuta. Este ejemplo ilustra el uso del ciclo **while:**

```
integer count;
initial
   begin
     count = 0;
     while (count < 64)
     count = count + 1;
   end
```

El valor de count se incrementa de 0 a 63. El ciclo termina cuando el conteo llega a 64.

Al manejar enunciados cíclicos, a veces resulta conveniente usar un tipo de datos entero para manipular cantidades. Los enteros se declaran con la palabra clave **integer,** como en el ejemplo anterior. Aunque es posible usar la palabra clave **reg** para declarar variables, si la variable se usará para contar es más recomendable declararla como **integer.** Las variables que se declaran como **reg** se almacenan como números sin signo. Las que se declaran como **integer** se almacenan como números con signo en formato de complemento a dos. La anchura por omisión de un entero es de 32 bits.

El ciclo **for** tiene tres partes, separadas por signos de punto y coma:

- Una condición inicial.
- Una expresión para verificar la condición de terminación.
- Una asignación para modificar la variable de control.

Ejemplo HDL 8-1

```
//descripción de un decodificador 2x4
//empleando el enunciado cíclico for
module decoder (IN, Y);
   input  [1:0] IN;    //Dos entradas binarias
   output [3:0] Y;    //Cuatro salidas binarias
   reg [3:0] Y;
   integer I;          //Variable de control del ciclo
   always @ (IN)
     for (I = 0; I <= 3; I = I + 1)
       if (IN == I) Y[I] = 1;
       else Y[I] = 0;
endmodule
```

He aquí un ejemplo de ciclo **for:**

```
for(i = 0; i < 8; i + 1)
enunciados procedimentales
```

El enunciado cíclico repite ocho veces la ejecución de los enunciados procedimentales. La variable de control es *i*, la condición inicial es $i = 0$, el ciclo se repite en tanto *i* sea menor que 8. Cada vez que el ciclo se ejecuta, *i* se incrementa en 1.

En el ejemplo HDL 8-1 se muestra la descripción de un decodificador de 2 a 4 líneas empleando un enunciado **for.** Puesto que la salida *Y* se evalúa en un enunciado procedimental, se le debe declarar como de tipo **reg.** La variable de control del ciclo es el **integer** I. Si expandimos el ciclo, obtendremos estas cuatro condiciones (*IN* y *Y* están en binario, el índice de *Y* está en decimal):

$$\text{if} \quad IN = 00 \text{ then } Y(0) = 1 \quad \text{else } Y(0) = 0$$
$$\text{if} \quad IN = 01 \text{ then } Y(1) = 1 \quad \text{else } Y(1) = 0$$
$$\text{if} \quad IN = 10 \text{ then } Y(2) = 1 \quad \text{else } Y(2) = 0$$
$$\text{if} \quad IN = 11 \text{ then } Y(3) = 1 \quad \text{else } Y(3) = 0$$

Síntesis lógica

Síntesis lógica es el proceso automático de transformar una descripción, escrita en un lenguaje de alto nivel como HDL, en una lista optimizada de una red de compuertas que efectúa las operaciones especificadas por el código fuente. Hay diversas tecnologías que implementan el diseño sintetizado. Para poder utilizar eficazmente una descripción HDL, el diseñador debe adoptar un estilo apropiado para las herramientas de síntesis que usa. El tipo de CI que implementa el diseño podría ser un circuito integrado para una aplicación específica (ASIC, *application-specific integrated circuit*), un dispositivo lógico programable (PLD) o un arreglo de compuertas programable en el campo (FPGA).

Las herramientas de síntesis lógica son programas que interpretan el código fuente escrito en lenguaje de descripción de hardware y lo traducen a una estructura de compuertas. Los diseños escritos en HDL con fines de síntesis lógica suelen especificarse en el nivel de transferencia de registros (RTL). El motivo es que las construcciones HDL empleadas en una descripción RTL se pueden convertir directamente en una descripción en el nivel de com-

puertas. Los ejemplos que siguen muestran cómo un sintetizador lógico puede interpretar una construcción HDL y convertirla en una estructura de compuertas.

El enunciado **assign** sirve para describir los circuitos combinacionales. Un enunciado **assign** con ecuaciones booleanas se interpreta para dar el circuito de compuertas correspondiente. Un enunciado con signo más (+) se interpreta como sumador binario con circuitos de sumador completo. Un enunciado con signo menos (−) se convierte en un restador que consiste en sumadores completos y compuertas OR exclusivo (figura 4-13). Un enunciado con un operador condicional, como

```
assign Y = S ? I1 : I0;
```

se traduce en un multiplexor de 2 líneas a 1, con entrada de control S y entradas de datos I1 e I0. Un enunciado con varios operadores condicionales especifica un multiplexor más grande.

El enunciado **always** podría implicar un circuito combinacional o un circuito secuencial. En el caso de los circuitos secuenciales, el control de sucesos debe ser **posedge** o **negedge** de un reloj; de lo contrario, el enunciado procedimental especificará un circuito combinacional. Por ejemplo,

```
always @ (I1 or I0 or S)
      if (S) Y = I1;
      else   Y = I0;
```

se traduce en un multiplexor de 2 líneas a 1. Se puede usar el enunciado **case** para implicar multiplexores grandes.

El enunciado **always @ posedge** o **negedge** reloj especifica circuitos secuenciales con reloj. Los circuitos correspondientes consisten en flip-flops *D* y las compuertas que implementan las operaciones de transferencia de registros. Los registros y contadores son ejemplos de tales circuitos. Una descripción de circuito secuencial con un enunciado **case** se traduce en un circuito de control con flip-flops *D* y compuertas. Así pues, el sintetizador interpreta cada enunciado de una descripción RTL y lo asigna a un circuito de compuertas y flip-flops correspondientes.

La figura 8-1 corresponde a un diagrama de flujo simplificado del proceso de diseño. La descripción RTL del diseño HDL se simula, verificándose su correcto funcionamiento. El conjunto de pruebas genera las señales de estímulo del simulador. Si el resultado de la simulación no es satisfactorio, la descripción HDL se corrige y se vuelve a verificar. Una vez que el resultado de la simulación indica que el diseño es válido, la descripción RTL se aplica al sintetizador lógico. Las herramientas de síntesis generan la lista de una red equivalente a una descripción del diseño en el nivel de compuertas. El circuito a nivel de compuertas se simula con el mismo conjunto de estímulos que se usaron para verificar el diseño RTL. Si se requieren correcciones, el proceso se repite hasta lograrse una simulación satisfactoria. Los resultados de las dos simulaciones se comparan para ver si coinciden. Si no coinciden, el diseñador modifica la descripción RTL, corrigiendo los defectos del diseño. Luego se introduce otra vez la descripción en el sintetizador lógico para generar una nueva descripción en el nivel de compuertas. Una vez que el diseñador queda satisfecho con los resultados de todas las pruebas de simulación, el circuito ya puede fabricarse con un circuito integrado.

La síntesis lógica ofrece varias ventajas al diseñador. Es menos tardado escribir una descripción HDL y sintetizar una realización en el nivel de compuertas, que desarrollar el circuito capturando manualmente diagramas esquemáticos. La facilidad para modificar la descripción facilita la exploración de alternativas de diseño. Es más fácil comprobar la validez del diseño por simulación, que producir un prototipo en hardware para evaluación. Las herramientas de síntesis permiten generar automáticamente la base de datos necesaria para fabricar el circuito integrado.

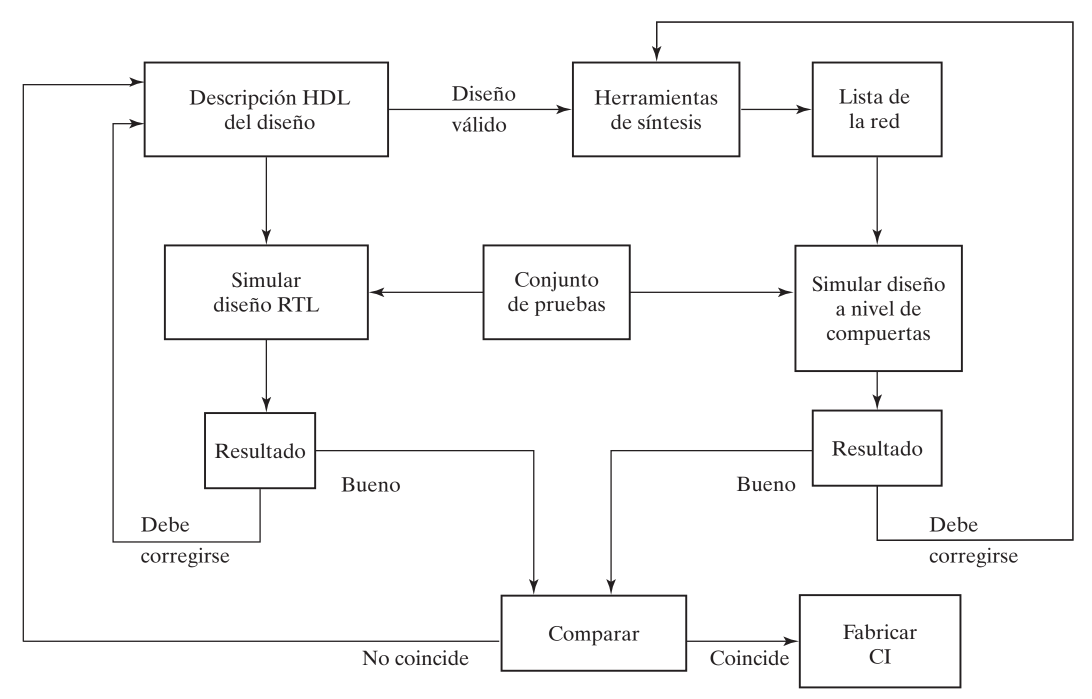


FIGURA 8-1
Proceso de simulación y síntesis en HDL

8-3 MÁQUINAS DE ESTADOS ALGORÍTMICAS

La información binaria almacenada en un sistema digital se clasifica como datos o información de control. Los datos son elementos discretos de información que se manipulan para llevar a cabo tareas de procesamiento de datos como aritmética, lógica, desplazamiento, etcétera. Estas operaciones se implementan con componentes digitales como sumadores, decodificadores, multiplexores, contadores y registros de desplazamiento. La información de control suministra señales de mando que supervisan las diversas operaciones en la sección de datos para que se efectúen las tareas de procesamiento deseadas. El diseño lógico de un sistema digital se divide en dos partes bien definidas. Una se ocupa de diseñar los circuitos digitales que efectúan las operaciones de procesamiento de datos. La otra se ocupa de diseñar los circuitos de control que determinan el orden en que se efectuarán las diversas acciones.

La relación entre la lógica de control y el procesamiento de datos en un sistema digital se indica en la figura 8-2. La trayectoria de procesamiento de datos, llamada comúnmente *trayectoria de datos*, manipula datos en registros según los requisitos del sistema. La lógica de control inicia el envío de una sucesión de órdenes a la trayectoria de datos. La lógica de control utiliza condiciones de estado de la trayectoria de datos como variables de decisión para determinar el orden de las señales de control.

La lógica de control que genera las señales que indican el orden de las operaciones en la trayectoria de datos es un circuito secuencial cuyos estados internos determinan las órdenes de control del sistema. En cualquier momento dado, el estado del control secuencial genera un conjunto

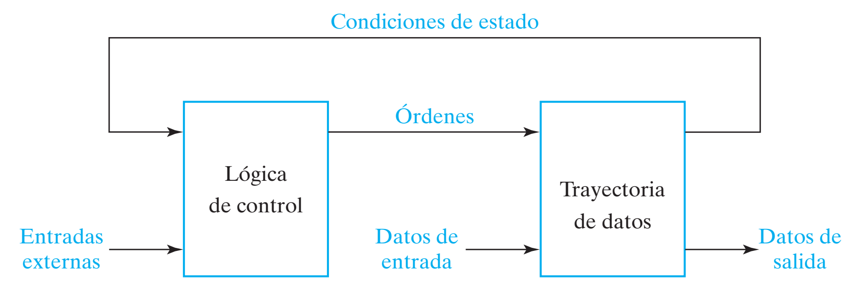


FIGURA 8-2
Interacción del control y la trayectoria de datos

prescrito de órdenes. Dependiendo de condiciones de estado y otras entradas externas, el control secuencial pasa al siguiente estado para iniciar otras operaciones. Los circuitos digitales que actúan como lógica de control suministran una secuencia temporal de señales para iniciar las operaciones en la trayectoria de datos, y también determinan el siguiente estado del subsistema de control mismo.

La secuencia de control y las tareas de la trayecotoria de datos de un sistema digital se especifican mediante un algoritmo en hardware. Un algoritmo es un número finito de pasos procedimentales que especifican cómo resolver un problema. Un algoritmo en hardware es un procedimiento para implementar la solución al problema con un equipo dado. La parte más difícil y creativa del diseño digital es la formulación de algoritmos en hardware para lograr los objetivos deseados.

El diagrama de flujo es una forma conveniente de especificar la serie de pasos procedimentales y trayectorias de decisión de un algoritmo. Un diagrama de flujo para un algoritmo en hardware traduce el planteamiento en palabras en un diagrama de información que enumera la serie de operaciones, junto con las condiciones necesarias para su ejecución. Un diagrama de flujo especial que se creó específicamente para definir algoritmos digitales en hardware se llama diagrama de *máquina de estados algorítmica* (ASM, *algorithmic state machine*). *Máquina de estados* es otro nombre para un circuito secuencial, que es la estructura básica de un sistema digital.

El diagrama ASM se parece a los diagramas de flujo convencionales, pero se interpreta de forma un tanto distinta. Los diagramas convencionales describen la sucesión de pasos procedimentales y trayectorias de decisión de un algoritmo en forma secuencial, sin tomar en cuenta sus relaciones temporales. El diagrama ASM describe la serie de sucesos y también las relaciones de temporización entre los estados del controlador secuencial y los sucesos que tienen lugar cuando pasa de un estado al siguiente. Está adaptado específicamente para describir con exactitud la sucesión de control y las operaciones de trayectoria de datos de un sistema digital, tomando en consideración las restricciones del hardware digital.

Diagrama ASM

El diagrama ASM es un tipo especial de diagrama de flujo apropiado para describir las operaciones secuenciales de un sistema digital. El diagrama se compone de tres elementos básicos: el cuadro de estado, el cuadro de decisión y el cuadro condicional. Un estado de la sucesión de control se indica con un cuadro de estado, como se observa en la figura 8-3. La forma del

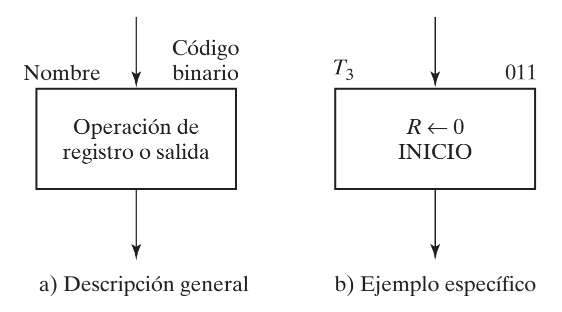


FIGURA 8-3
Cuadro de estado

cuadro es rectangular, y en su interior se escriben operaciones de registros o nombres de señales de salida que el control genera cuando está en ese estado. El estado recibe un nombre simbólico, que se coloca en la esquina superior izquierda del cuadro. El código binario asignado al estado se coloca en la esquina superior derecha. La figura 8-3b) muestra un ejemplo específico de cuadro de estado. El estado tiene el nombre simbólico T_3, y se le ha asignado el código binario 011. Dentro del cuadro se ha escrito la operación de registro $R \leftarrow 0$, que indica que el registro R debe ponerse en ceros cuando el sistema está en el estado T_3. El nombre INICIO dentro del cuadro podría indicar, por ejemplo, una señal de salida que inicia cierta operación.

El cuadro de decisión describe el efecto de una entrada sobre el subsistema de control. Tiene forma de rombo con dos o más trayectorias de salida, como se aprecia en la figura 8-4. La condición de entrada que se probará se escribe dentro del rombo. Se toma una trayectoria de salida si la condición se cumple, y el otro cuando no se cumple. Si se asigna un valor binario a una condición de entrada, las dos trayectorias se marcan con 1 y 0.

Los cuadros de estado y decisión son conocidos porque se usan en los diagramas de flujo convencionales. El tercer elemento, el cuadro condicional, es exclusivo de los diagramas ASM. La forma ovalada del cuadro condicional se aprecia en la figura 8-5. Las esquinas redondeadas lo distinguen del cuadro de estado. La trayectoria de entrada al cuadro condicional debe provenir de una de las trayectorias de salida de un cuadro de decisión. Las operaciones de registros o salidas que se anotan dentro del cuadro condicional se generan durante un estado dado, a condición de que se satisfaga la condición de entrada. La figura 8-5b) presenta un ejemplo con cuadro condicional. El control genera una señal de salida INICIO cuando está en el estado T_1. Mientras está en ese estado, el control verifica la situación de la entrada E. Si $E = 1$, R se pone en ceros; en caso contrario, R no cambia. En ambos casos, el siguiente estado es T_2.

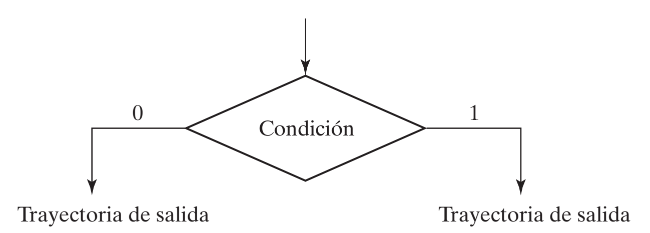


FIGURA 8-4
Cuadro de decisión

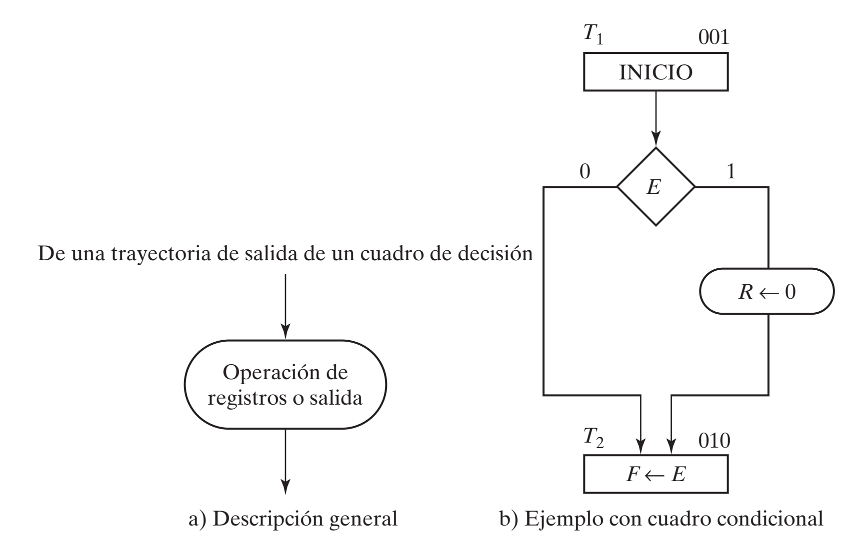


FIGURA 8-5
Cuadro condicional

Bloque ASM

Un bloque ASM es una estructura que consiste en un cuadro de estado y todos los cuadros de decisión y condicionales conectados a su trayectoria de salida. Cada bloque ASM tiene una sola entrada y cualquier cantidad de trayectorias de salida, representados por la estructura de los cuadros de decisión. Un diagrama ASM consiste en uno o más bloques interconectados. En la figura 8-6 se ilustra un ejemplo de bloque ASM. El estado T_1 está asociado a dos cuadros de decisión y un cuadro condicional. El diagrama distingue el bloque con líneas punteadas que encierran toda la estructura, pero no se acostumbra hacer esto porque el diagrama ASM

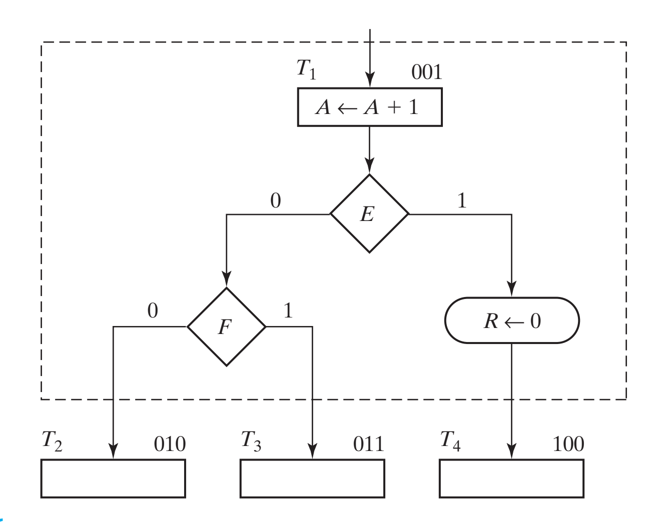


FIGURA 8-6
Bloque ASM

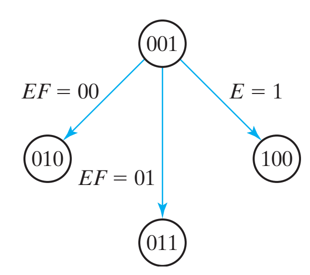


FIGURA 8-7
Diagrama de estados equivalente al diagrama ASM de la figura 8-6

define de forma única cada bloque a partir de su estructura. Un cuadro de estado sin cuadros de decisión ni condicionales constituye un bloque simple.

Cada bloque del diagrama ASM describe el estado del sistema durante un intervalo de pulso de reloj. Las operaciones especificadas dentro de los cuadros de estado y condicionales de la figura 8-6 se ejecutan con un pulso de reloj común mientras el sistema está en el estado T_1. El mismo pulso de reloj transfiere el controlador del sistema a uno de los siguientes estados —T_2, T_3 o T_4— según determinen los valores binarios de E y F.

El diagrama ASM es muy similar a un diagrama de estados. Cada cuadro de estado equivale a un estado de un circuito secuencial. El cuadro de decisión equivale a la información binaria que se escribe sobre las flechas que conectan dos estados de un diagrama de estados. Por ello, hay ocasiones en que conviene convertir el diagrama ASM en un diagrama de estados y luego usar procedimientos de circuitos secuenciales para diseñar la lógica de control. Para ilustrar esto, se ha dibujado el diagrama ASM de la figura 8-6 como diagrama de estados en la figura 8-7. Los tres estados se indican con círculos, con su valor binario escrito adentro. Las flechas indican las condiciones que determinan el siguiente estado. Las operaciones incondicionales y condicionales que deben efectuarse no se indican en el diagrama de estados.

Consideraciones de temporización

La temporización de todos los registros y flip-flops de un sistema digital se controla con un generador de reloj maestro. Los pulsos de reloj se aplican no sólo a los registros de la trayectoria de datos, sino también a todos los flip-flops de la lógica de control. Las entradas también se sincronizan con el reloj porque normalmente se generan como salidas de otro circuito que usa las mismas señales de reloj. Si la señal de entrada cambia en un momento arbitrario, independiente del reloj, decimos que es una entrada asincrónica. Las entradas asincrónicas pueden causar diversos problemas, como se verá en el capítulo 9. Para simplificar el diseño, supondremos que todas las entradas están sincronizadas con el reloj y cambian de estado en respuesta a una transición de borde.

La principal diferencia entre un diagrama de flujo convencional y un diagrama ASM es la interpretación de la relación temporal entre las diversas operaciones. Por ejemplo, si la figura 8-6 fuera un diagrama de flujo convencional, se consideraría que las operaciones indicadas se siguen una a la otra en orden temporal: primero se incrementa el registro A y luego se evalúa E. Si $E = 1$, el registro R se pone en ceros y el control pasa al estado T_4. De lo contrario, si $E = 0$, el siguiente paso es evaluar F y pasar al estado T_2 o al T_3. En contraste, un diagrama ASM considera a todo el bloque como una unidad. Todas las operaciones que se especifican

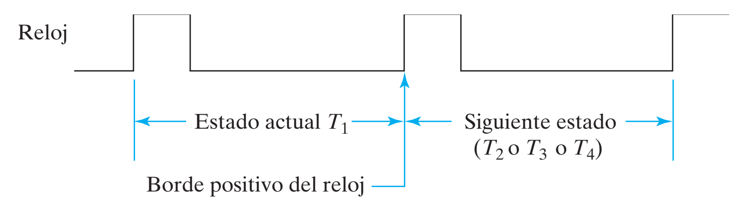


FIGURA 8-8
Transición entre estados

dentro del bloque deben ejecutarse en sincronía durante la transición de borde del mismo pulso de reloj mientras el sistema cambia de T_1 al siguiente estado. Esto se presenta de forma gráfica en la figura 8-8. Se supone que todos los flip-flops se disparan con el borde positivo. La primera transición positiva del reloj transfiere el circuito de control al estado T_1. Mientras está en el estado T_1, los circuitos de control examinan las entradas E y F y generan las señales apropiadas, según sus valores. Las operaciones que siguen se ejecutan simultáneamente durante el siguiente borde positivo del reloj:

1. El registro A se incrementa.
2. Si $E = 1$, el registro R se despeja.
3. Se transfiere el control al siguiente estado según especifica la figura 8-7.

Tome en cuenta que las dos operaciones en el camino de datos y el cambio de estado en la lógica de control suceden al mismo tiempo.

8-4 EJEMPLO DE DISEÑO

Ahora ilustraremos los componentes del diagrama ASM y la representación de transferencia de registros examinando un ejemplo específico de diseño. Partiremos de las especificaciones iniciales y procederemos con el desarrollo de un diagrama ASM apropiado, con base en el cual se diseñará después el hardware digital.

Queremos diseñar un sistema digital con dos flip-flops, E y F, y un contador binario de cuatro bits, A. Los flip-flops individuales de A se designarán con A_4, A_3, A_2 y A_1, donde A_4 contiene el bit más significativo de la cuenta. Una señal de inicio S pone en marcha el funcionamiento del sistema despejando el contador A y el flip-flop F. Luego el contador se incrementa en uno a partir del siguiente pulso de reloj y se sigue incrementando hasta que el sistema para. Los bits A_3 y A_4 del contador determinan la sucesión de operaciones:

Si $A_3 = 0$, E se pone en 0 y el conteo continúa.

Si $A_3 = 1$, E se pone en 1; entonces, si $A_4 = 0$, el conteo continúa, pero si $A_4 = 1$, F se pone en 1 en el siguiente pulso de reloj y el sistema deja de contar.

Entonces, si $S = 0$, el sistema permanece en el estado inicial, pero si $S = 1$, se repite el ciclo de funcionamiento.

Diagrama ASM

El diagrama ASM se reproduce en la figura 8-9. Cuando no se efectúan operaciones, el sistema está en el estado inicial T_0, en espera de la señal de inicio S. Cuando la entrada S es 1, el contador A y el flip-flop F se ponen en ceros y el controlador pasa al estado T_1. Observe el cuadro condicional que sigue al cuadro de decisión de S. Esto implica que el contador y el flip-flop se despejarán durante T_0 si $S = 1$, y al mismo tiempo se transferirá el control al estado T_1.

El bloque asociado al estado T_1 tiene dos cuadros de decisión y dos cuadros condicionales. El contador se incrementa con cada pulso de reloj. Al mismo tiempo, se efectúa una de tres posibles operaciones durante el mismo borde de reloj:

E se despeja y el control permanece en el estado T_1 ($A_3 = 0$); o bien,
E se pone en 1 y el control permanece en el estado T_1 ($A_3A_4 = 10$); o
E se pone en 1 y el control pasa al estado T_2 ($A_3A_4 = 11$).

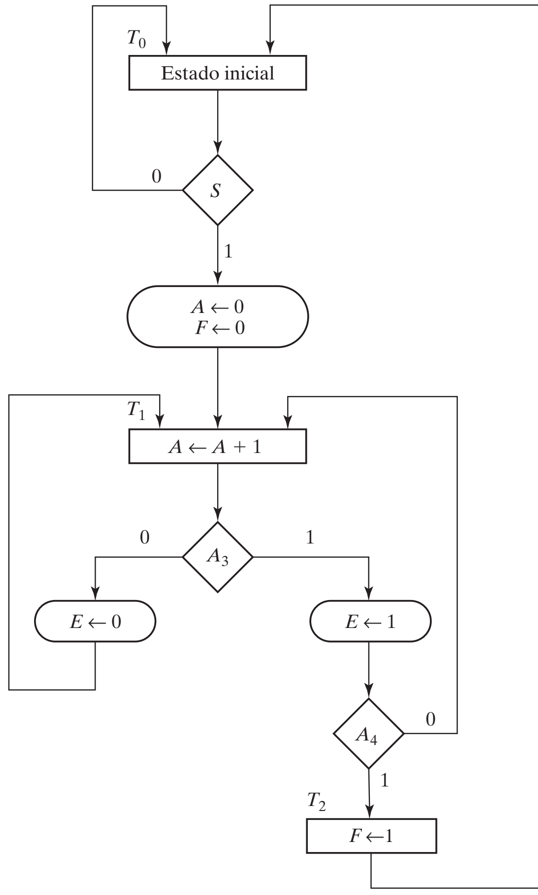


FIGURA 8-9
Diagrama ASM para el ejemplo de diseño

Cuando el control está en el estado T_2, el flip-flop F se pone en 1 y el circuito vuelve al estado inicial, T_0.

El diagrama ASM consta de tres estados y tres bloques. El bloque asociado a T_0 consiste en el cuadro de estado, un cuadro de decisión y un cuadro condicional. El bloque asociado a T_2 consiste únicamente en el cuadro de estado. La lógica de control tiene una entrada externa, S, y dos entradas de situación, A_3 y A_4.

Hemos mostrado aquí la forma en que una descripción textual de un diseño se traduce a un diagrama ASM. Hay que tener presente que el ejemplo de diseño formulado en el diagrama ASM no tiene una aplicación práctica y, dependiendo de la interpretación, podría simplificarse y formularse de otra manera. Sin embargo, una vez establecido el diagrama ASM, el procedimiento para diseñar el circuito es directo.

Secuencia de temporización

Cada bloque de un diagrama ASM especifica las operaciones que se efectuarán durante un pulso de reloj común. Las operaciones especificadas dentro de los cuadros de estado y condicionales del bloque se ejecutan en la subsección de trayectoria de datos. El cambio de un estado al siguiente se efectúa en la lógica de control. Para apreciar la relación de temporización que interviene, numeraremos la sucesión de operaciones que se efectúan después de cada pulso de reloj, desde el momento en que se presenta la señal de inicio hasta que el sistema vuelve al estado inicial.

La tabla 8-2 indica los valores binarios del contador y los dos flip-flops después de cada pulso de reloj. La tabla también muestra, por separado, la situación de A_3 y A_4, así como el estado actual del controlador. Iniciamos con el estado T_1 inmediatamente después de que la señal de entrada S ha hecho que el contador y el flip-flop F se pongan en ceros. Se supone que el valor de E es 1, porque E es 1 en T_0 (como se indica al final de la tabla) y porque E no cambia durante la transición de T_0 a T_1. El sistema permanece en el estado T_1 durante los siguientes 13 pulsos de reloj. Cada pulso incrementa el contador y apaga o bien enciende E. Note la relación entre el momento en que A_3 se vuelve 1 y el momento en que E se pone en 1. Cuando $A = 0011$, el siguiente pulso de reloj incrementa el contador a 0100, pero ese mismo borde de reloj ve un 0 en A_3, así que despeja E. El siguiente pulso cambia el contador de 0100 a 0101, y ahora A_3 es inicialmente 1, así que E se pone en 1. De forma similar, E se pone en 0 no cuando la cuenta pasa de 0111 a 1000, sino cuando pasa de 1000 a 1001, que es cuando A_3 es 0 en el valor actual del contador.

Cuando la cuenta llega a 1100, tanto A_3 como A_4 son 1. El siguiente borde de reloj incrementa A en 1, pone E en 1 y transfiere el control al estado T_2. El control permanece en T_2 durante un solo periodo de reloj. El borde de reloj asociado a T_2 pone en 1 el flip-flop F y transfiere el control al estado T_0. El sistema permanecerá en el estado inicial T_0 en tanto S sea 0.

Al examinar la tabla 8-2, podría parecer que las operaciones que se efectúan sobre E se retardan un pulso de reloj. Ésta es la diferencia entre un diagrama ASM y un diagrama de flujo convencional. Si la figura 8-9 fuera un diagrama de flujo convencional, supondríamos que primero se incrementa A y luego se usa el valor incrementado para verificar la situación de A_3. Las operaciones que se efectúan en el hardware digital especificado por un bloque del diagrama ASM tienen lugar durante el mismo ciclo de reloj y no como una sucesión de operaciones que se efectúan una tras otra en el tiempo, que es como suelen interpretarse los diagramas de flujo convencionales. Así pues, el valor de A_3 que se considera en el cuadro de decisión se toma del valor del contador en el estado actual, antes de incrementarse. El motivo es que el cuadro

Tabla 8-2
Sucesión de operaciones para el ejemplo de diseño

Contador				Flip-flops			
A_4	A_3	A_2	A_1	E	F	**Condiciones**	**Estado**
0	0	0	0	1	0	$A_3 = 0, A_4 = 0$	T_1
0	0	0	1	0	0		
0	0	1	0	0	0		
0	0	1	1	0	0		
0	1	0	0	0	0	$A_3 = 1, A_4 = 0$	
0	1	0	1	1	0		
0	1	1	0	1	0		
0	1	1	1	1	0		
1	0	0	0	1	0	$A_3 = 0, A_4 = 1$	
1	0	0	1	0	0		
1	0	1	0	0	0		
1	0	1	1	0	0		
1	1	0	0	0	0	$A_3 = 1, A_4 = 1$	
1	1	0	1	1	0		T_2
1	1	0	1	1	1		T_0

de decisión de E pertenece al mismo bloque que el estado T_1. Los circuitos digitales del control generan las señales para todas las operaciones especificadas en el bloque actual, antes de la llegada del siguiente pulso de reloj. El siguiente borde de reloj ejecuta todas las operaciones de los registros y flip-flops, incluidos los flip-flops del controlador que determinan el siguiente estado.

Diseño de trayectorias de datos

El diagrama ASM proporciona toda la información necesaria para diseñar el sistema digital. Los requisitos para el diseño de la trayectoria de datos se especifican dentro de los cuadros de estado y condicionales. La lógica de control se determina a partir de los cuadros de decisión y de las transiciones de estado requeridas. La figura 8-10 es un diagrama que muestra el hardware para el ejemplo de diseño. El subsistema de control se exhibe sólo con sus entradas y salidas. El diseño detallado del control se considerará más adelante. La trayectoria de datos consiste en un contador binario de cuatro bits, dos flip-flops y varias compuertas. El contador es similar al que se ilustra en la figura 6-12, excepto que se requieren compuertas adicionales para la operación sincrónica de despeje. El contador se incrementa con cada ciclo de reloj cuando el control está en el estado T_1. Sólo se despeja cuando el control está en el estado T_0 y S es 1. Esta operación condicional requiere una compuerta AND para garantizar que ambas condiciones estén presentes. Las otras dos operaciones condicionales utilizan otras dos compuertas AND para encender o apagar el flip-flop E. El flip-flop F se enciende incondicionalmente durante el estado T_2. Todos los flip-flops y registros, incluidos los flip-flops del control, usan el mismo reloj.

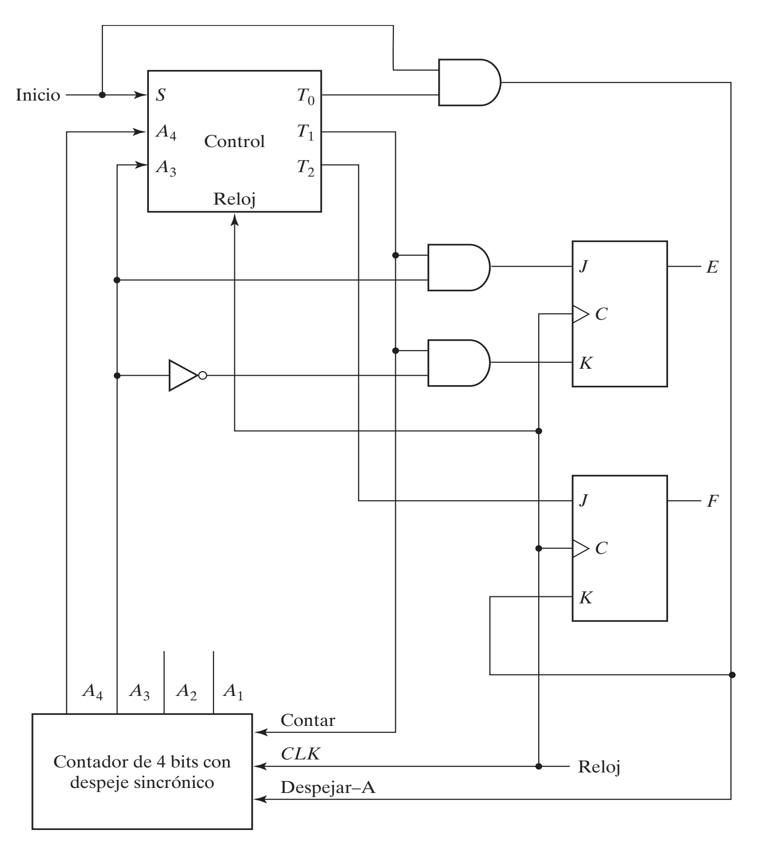
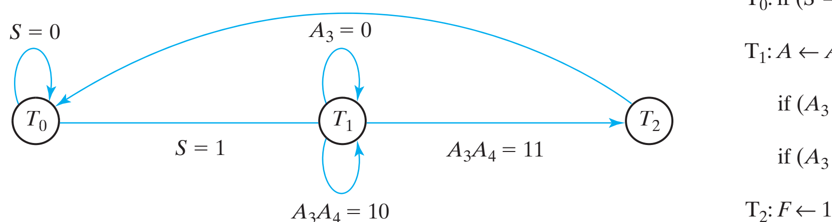


FIGURA 8-10
Trayectoria de datos del ejemplo de diseño

Representación de transferencia de registros

Los sistemas digitales se representan en el nivel de transferencia de registros especificando los registros del sistema, las operaciones efectuadas y la sucesión de control. Las operaciones de registro y la información de control se especifican con un diagrama ASM. Hay ocasiones en que conviene separar la lógica de control y las operaciones de registros para la trayectoria de datos. La información de control y las operaciones de transferencia de registros se representan por separado como se aprecia en la figura 8-11. El diagrama de estados especifica la sucesión de control, y las operaciones de registros se especifican con la notación que se presenta en la sección 8-1. Esta representación es una alternativa a la representación del sistema descrita en el diagrama ASM de la figura 8-9. La información para el diagrama de estados se tomó directamente del diagrama ASM. Los nombres de los estados se especifican en cada cuadro de estado. Las condiciones que causan un cambio de estado se especifican dentro de los cuadros de decisión rómbicos. Las flechas entre los estados y la condición asociada a cada una siguen la misma trayectoria que en el diagrama ASM. Las operaciones de transferencia de registros para cada uno de los tres estados se numeran después del nombre del estado y de un signo de dos puntos (:). Se toman de los cuadros de estado rectangulares y de los cuadros condicionales ovalados correspondientes del diagrama ASM.



a) Diagrama de estados para el control

T_0: if $(S = 1)$ then $A \leftarrow 0, F \leftarrow 0$

T_1: $A \leftarrow A + 1$

if $(A_3 = 1)$ then $E \leftarrow 1$

if $(A_3 = 0)$ then $E \leftarrow 0$

T_2: $F \leftarrow 1$

b) Operaciones de transferencia de registros

FIGURA 8-11
Descripción del ejemplo de diseño en el nivel de transferencia de registros

Tabla de estados

El diagrama de estados se puede convertir en una tabla de estados con base en la cual es posible diseñar el circuito secuencial del controlador. Primero, debemos asignar valores binarios a cada estado del diagrama ASM. Para n flip-flops en el circuito secuencial de control, el diagrama ASM da cabida hasta a 2^n estados. Un diagrama con tres o cuatro estados requiere un circuito secuencial con dos flip-flops. Con cinco a ocho estados, se necesitan tres flip-flops. Cada combinación de valores de los flip-flops representa un número binario para uno de los estados.

La tabla de estados de un controlador es una lista de estados actuales y entradas, y sus correspondientes siguientes estados y salidas. En la mayoría de los casos, hay muchas condiciones de entrada indiferentes que es preciso incluir, por lo que es aconsejable acomodar la tabla de estados tomando eso en cuenta. Asignamos los valores binarios siguientes a los tres estados: $T_0 = 00$, $T_1 = 01$, $T_2 = 11$. El estado binario 10 no se usa y se tratará como condición de indiferencia. La tabla de estados correspondiente al diagrama de estados aparece en la tabla 8-3. Se requieren dos flip-flops, que se han rotulado G_1 y G_0. Hay tres entradas y tres salidas. Las entradas se toman de las condiciones que están en los cuadros de decisión. Las salidas son equivalentes al estado actual del control. Vemos que en la tabla hay una fila para cada posible transición entre estados. El estado inicial 00 pasa al estado 01 o se queda en 00, dependiendo del valor de la entrada S. Las otras dos entradas se marcan con la X que denota indiferencia,

Tabla 8-3
Tabla de estados para el control de la figura 8-10

Símbolo del estado actual	Estado actual		Entradas			Siguiente estado		Salidas		
	G_1	G_0	S	A_3	A_4	G_1	G_0	T_0	T_1	T_2
T_0	0	0	0	X	X	0	0	1	0	0
T_0	0	0	1	X	X	0	1	1	0	0
T_1	0	1	X	0	X	0	1	0	1	0
T_1	0	1	X	1	0	0	1	0	1	0
T_1	0	1	X	1	1	1	1	0	1	0
T_2	1	1	X	X	X	0	0	0	0	1

ya que no determinan el siguiente estado en este caso. Mientras el sistema está en el estado binario 00, el control proporciona una salida rotulada T_0 que inicia las operaciones de registros requeridas. La transición desde el estado binario 01 depende de las entradas A_3 y A_4. El sistema pasa al estado binario 11 sólo si $A_3A_4 = 11$; de lo contrario, permanece en el estado binario 01. Por último, el estado binario 11 pasa a 00 independientemente de las variables de entrada.

Lógica de control

El procedimiento para diseñar un circuito secuencial a partir de una tabla de estados se presentó en el capítulo 5. Si se aplica ese procedimiento a la tabla 8-3, se necesitará usar mapas de cinco variables para simplificar las ecuaciones de entrada. El motivo es que aparecen cinco variables bajo las columnas de estado actual y entradas de la tabla. En lugar de usar mapas para simplificar las ecuaciones de entrada, podemos obtenerlas directamente de la tabla de estados por inspección. Si queremos diseñar el circuito secuencial con flip-flops D, habrá que estudiar las columnas de siguiente estado de la tabla y deducir todas las condiciones que deben poner en 1 cada flip-flop. En la tabla 8-3 vemos que la columna de siguiente estado para G_1 tiene un solo 1 en la quinta fila. La entrada D del flip-flop G_1 debe ser 1 durante el estado actual T_1, cuando ambas entradas, A_3 y A_4, son también 1. Esto se expresa con la ecuación de entrada de flip-flop D.

$$D_{G1} = T_1A_3A_4$$

Asimismo, la columna de siguiente estado de G_0 tiene cuatro unos, y la condición para encender este flip-flop es

$$D_{G0} = T_0S + T_1$$

Para deducir las tres funciones de salida, podemos aprovechar el hecho de que el estado binario 10 no se usa, y obtener este conjunto simplificado de ecuaciones de salida:

$$T_0 = G_0'$$
$$T_1 = G_1'G_0$$
$$T_2 = G_1$$

El diagrama lógico del control se ha dibujado en la figura 8-12. Este circuito muestra la construcción interna del bloque de control de la figura 8-10 junto con la compuerta AND que genera la señal Despejar-A.

8-5 DESCRIPCIÓN DEL EJEMPLO DE DISEÑO EN HDL

En capítulos anteriores, se han incluido ejemplos de descripciones HDL de circuitos combinacionales, circuitos secuenciales y diversos componentes estándar como multiplexores, contadores y registros. Ya estamos en condiciones de incorporar estos componentes a una descripción de un diseño específico. Como se mencionó antes, los diseños se pueden describir en un nivel estructural o en un nivel de comportamiento. Las descripciones de comportamiento se efectúan en el nivel de transferencia de registros o en un nivel algorítmico abstracto. Por tanto, ahora consideraremos tres niveles de diseño: descripción estructural, descripción RTL y descripción de comportamiento basada en algoritmos.

La descripción *estructural* es el nivel más bajo y más detallado. El sistema digital se especifica en términos de los componentes físicos y de su interconexión. Los diversos componen-

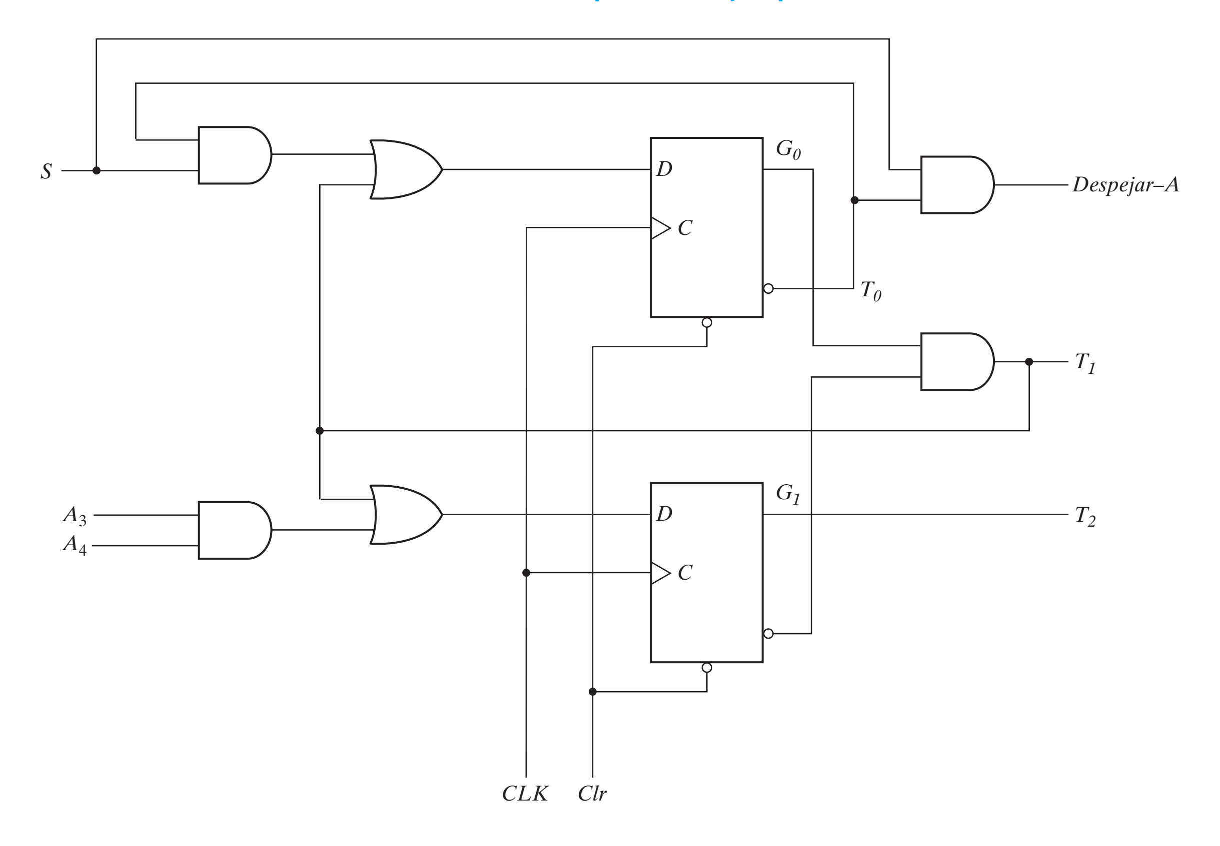


FIGURA 8-12
Diagrama lógico de control

tes podrían incluir compuertas, flip-flops y circuitos estándar, como multiplexores y contadores. El diseño se descompone jerárquicamente en unidades funcionales, cada una de las cuales se describe con un módulo HDL. Un módulo en el nivel más alto combina todo el sistema creando ejemplares de todos los módulos de nivel inferior.

La descripción *RTL* especifica el sistema digital en términos de los registros, las operaciones efectuadas y el control que ordena las operaciones. Este tipo de descripción consiste en enunciados procedimentales que determinan la relación entre las diversas operaciones del diseño, sin referirlas a una estructura específica. La descripción RTL implica cierta configuración de hardware entre los registros. Esto permite al diseñador crear un diseño que puede sintetizarse para dar componentes digitales estándar.

La descripción *de comportamiento basada en algoritmos* es el nivel más abstracto. Describe la función del diseño en una forma algorítmica procedimental parecida a los lenguajes de programación. No da los pormenores de la implementación del diseño con hardware. Es la más apropiada para simular sistemas complejos con el fin de verificar las ideas de diseño. Las descripciones en este nivel son accesibles para usuarios no técnicos que saben de lenguajes de programación. Algunas de las construcciones en este nivel podrían no ser sintetizables.

A continuación ilustraremos las descripciones RTL y estructural utilizando el ejemplo de diseño de la sección anterior. En la sección 8-8 se ilustrará una descripción basada en algoritmos.

Descripción RTL

La descripción HDL de un diseño RTL se divide en tres secciones. La primera sección define las entradas, salidas y registros del diseño. La segunda sección especifica la sucesión de control. La tercera sección proporciona las operaciones de transferencia de registros y las salidas. La descripción RTL del ejemplo de diseño se muestra en el ejemplo HDL 8-2. Se ajusta al diagrama ASM de la figura 8-9 y a las especificaciones de transferencia de registros de la figura 8-11. Las entradas son S (inicio), CLK (reloj) y Clr (despejar). Esta última es necesaria para asignar $T0$ como estado inicial del control. Las salidas son los flip-flops E y F y el registro A. El registro de control se especifica con un vector de dos bits que contiene el valor actual de las condiciones de estado actual y siguiente estado. Los tres estados de control reciben nombres simbólicos y se codifican con valores binarios.

La siguiente sección de la descripción RTL especifica la sucesión de control con dos enunciados **always.** El primer bloque **always** incluye dos operaciones: la entrada Clr inicia el estado actual con $T0$, y CLK **posedge** sincroniza la transición de estado con el reloj. El segundo bloque **always** consiste en una condición **case** que especifica la transición del estado actual al siguiente estado. Por ejemplo, si el estado actual es $T0$ y $S = 1$, el siguiente estado será $T1$. Si $S = 0$, el estado $T0$ no cambiará. El cambio de $T0$ a $T1$ (si $S = 1$) sólo se efectúa cuando se presenta el suceso de borde positivo de CLK, como especifica el primer enunciado **always.**

La tercera sección de la descripción RTL especifica las operaciones de transferencia de registros en cada uno de los tres estados de control. Esto sigue las operaciones de transferencia de registros que se dan en la figura 8-11b). Advierta que se emplean asignaciones no bloqueadoras (con el símbolo <=) para las operaciones de transferencia de registros. Esto es crucial sobre todo durante el estado de control $T1$. En este estado, A se incrementa en uno y se verifica el valor de $A[3]$ para determinar la situación de E. Si se quiere lograr un diseño sincrónico válido, es preciso asegurarse de que $A[3]$ se verifique antes de incrementarse A. Si se usaran asignaciones bloqueadoras, sería necesario colocar primero los dos enunciados que verifican E y colocar al final el enunciado que incrementa A. En cambio, al usar asignaciones no bloqueadoras, logramos la sincronización requerida sin preocuparnos por el orden de los enunciados.

Prueba de la descripción de diseño

La sucesión de operaciones para el ejemplo de diseño se investigó en la sección anterior. La tabla 8-2 muestra los valores de E y F mientras se incrementa el registro A. Sería interesante idear una prueba para el circuito que verifique la validez de la descripción HDL. El conjunto de pruebas del ejemplo HDL 8-3 se encarga de ello. (El procedimiento para escribir conjuntos de pruebas se explicó en la sección 4-11.) El módulo de prueba genera señales para S, CLK y Clr, y verifica los resultados obtenidos de los registros A, E y F. En un principio, la señal Clr se pone en 0 para iniciar el control, y S y CLK se ponen en 0. En el tiempo $t = 5$, la señal Clr se inhabilita poniéndola en 1, la entrada S se habilita poniéndola en 1, y el reloj se repite durante 16 ciclos. El enunciado **$monitor** exhibe los valores de A, E y F cada 10 ns. La salida de la simulación se presenta en el ejemplo bajo el título "Simulation log" (bitácora de simulación). En un principio (tiempo $= 0$), los valores de los registros se desconocen, así que se marcan con el símbolo **x.** La primera transición positiva del reloj en tiempo $= 10$ despeja A y F, pero no afecta a E, de modo que E se desconoce en este momento. El resto de la tabla es idéntico a la tabla 8-2. Vemos que, puesto que S sigue siendo 1 en tiempo $= 160$, la última entrada de la tabla muestra que A y F se ponen en 0 y E no cambia, permaneciendo en 1. Esto sucede durante la segunda transición de $T0$ a $T1$.

Ejemplo HDL 8-2

```
//Descripción RTL del ejemplo de diseño (figura 8-11)
module Example_RTL (S,CLK,Clr,E,F,A);
//Especificar entradas y salidas
//Véase el diagrama de bloques de la figura 8-10
   input S,CLK,Clr;
   output E,F;
   output [4:1] A;
//Especificar registros del sistema
   reg [4:1] A;              //Registro A
   reg E, F;                 //Flip-flops E y F
   reg [1:0] pstate, nstate; //Registro de control
//Codificar los estados
   parameter T0 = 2'b00, T1 = 2'b01, T2 = 2'b11;
//Transición de estado para la lógica de control
//Véase el diagrama de estados de la figura 8-11a)
   always @(posedge CLK or negedge Clr)
      if (~Clr) pstate = T0;    //Estado inicial
      else pstate <= nstate;    //Operaciones con reloj
   always @ (S or A or pstate)
       case (pstate)
         T0: if(S) nstate = T1; else nstate = T0;
         T1: if(A[3] & A[4]) nstate = T2; else nstate = T1;
         T2: nstate = T0;
       endcase
//Operaciones de transferencia de registros
//Véase la lista de operaciones, figura 8-11b)
   always @(posedge CLK)
       case (pstate)
         T0: if(S)
              begin
                 A <= 4'b0000;
                 F <= 1'b0;
              end
         T1:
              begin
                 A <= A + 1'b1;
                 if (A[3]) E <= 1'b1;
                 else  E <= 1'b0;
              end
         T2:  F <= 1'b1;
      endcase
endmodule
```

Ejemplo HDL 8-3

```
//Conjunto de pruebas para el ejemplo de diseño
module test_design_example;
  reg S, CLK, Clr;
  wire [4:1] A;
  wire E, F;
//Crear ejemplar del ejemplo de diseño
   Example_RTL dsexp (S,CLK,Clr,E,F,A);
    initial
      begin
           Clr = 0;
           S = 0;
           CLK = 0;
        #5 Clr = 1; S = 1;
           repeat (32)
           begin
            #5 CLK = ~ CLK;
           end
      end
    initial
     $monitor("A = %b E = %b F = %b time = %0d", A,E,F,$time);
endmodule

Simulation log:

A = xxxx E = x F = x time = 0
A = 0000 E = x F = 0 time = 10
A = 0001 E = 0 F = 0 time = 20
A = 0010 E = 0 F = 0 time = 30
A = 0011 E = 0 F = 0 time = 40
A = 0100 E = 0 F = 0 time = 50
A = 0101 E = 1 F = 0 time = 60
A = 0110 E = 1 F = 0 time = 70
A = 0111 E = 1 F = 0 time = 80
A = 1000 E = 1 F = 0 time = 90
A = 1001 E = 0 F = 0 time = 100
A = 1010 E = 0 F = 0 time = 110
A = 1011 E = 0 F = 0 time = 120
A = 1100 E = 0 F = 0 time = 130
A = 1101 E = 1 F = 0 time = 140
A = 1101 E = 1 F = 1 time = 150
A = 0000 E = 1 F = 0 time = 160
```

Descripción estructural

La descripción RTL de un diseño consiste en enunciados procedimentales que determinan el comportamiento funcional del circuito digital. Es posible compilar este tipo de presentación con herramientas de síntesis HDL para obtener el circuito equivalente del diseño en el nivel de compuertas. También es factible describir el diseño por su estructura, en lugar de su función. La descripción estructural de un diseño consiste en la creación de ejemplares de componentes que definen la estructura de interconexión del circuito. En este sentido, una descripción estructural equivale a un diagrama esquemático o a un diagrama de bloques del circuito.

El diagrama de bloques de la figura 8-10 proporciona la información necesaria para la descripción estructural. Por conveniencia, el circuito se descompone en tres partes:

1. El bloque de control.
2. Los flip-flops E y F y las compuertas asociadas.
3. El contador con despeje sincrónico.

Se puede obtener una descripción jerárquica del diseño con una creación anidada de ejemplares de las tres partes.

El ejemplo HDL 8-4 muestra la descripción estructural del ejemplo de diseño. Consta de seis módulos que se separan en cuatro partes:

1. El primer módulo crea ejemplares de los tres componentes.
2. Los siguientes dos módulos describen el control y su flip-flop D.
3. Los siguientes dos módulos describen a E y F y a su flip-flop JK.
4. El último módulo describe el contador.

El primer módulo declara las entradas y salidas del circuito. La lista de puertos es idéntica a la que se usa en la descripción RTL. Las salidas no se declaran como de tipo **reg** aquí porque se declaran como **reg** en los módulos de nivel inferior. El módulo del nivel más alto encapsula todo el diseño al crear ejemplares de los tres componentes de nivel inferior. Algunos de los puertos de los módulos creados son las entradas o salidas del circuito. Otros puertos son entradas generadas a partir de otros módulos o salidas que se requieren para otros módulos. Por ejemplo, el módulo de control `ctl` tiene las entradas $A[3]$ y $A[4]$ que vienen de la salida $A[4:1]$ del módulo `counter ctr`. Las salidas $T1$ y $T2$ del módulo `ctl` se utilizan como entradas en el módulo *EF*.

El módulo de control describe el circuito de la figura 8-12. Las salidas de los dos flip-flops $G1$ y $G0$, y sus entradas $DG1$ y $DG0$ se declaran como del tipo de datos **wire.** $G1$ y $G0$ no pueden declararse como del tipo de datos **reg** porque son salidas del ejemplar de flip-flop D. $DG1$ y $DG0$ no se consideran de tipo **reg** porque se usan en enunciados de asignación continua. Los cinco enunciados **assign** especifican la parte combinacional del circuito. Hay dos ecuaciones de entrada de flip-flop y tres ecuaciones de salida. Las salidas de los flip-flops $G1$ y $G0$, y las ecuaciones de entrada $DG1$ y $DG0$ sustituyen a la salida Q y a la entrada D en los ejemplares de flip-flops. En el siguiente módulo se describe entonces el flip-flop D. El módulo EF sigue el mismo patrón para los dos flip-flops JK. Primero se obtienen las ecuaciones de entrada de flip-flop, y se crean los ejemplares de flip-flop JK con estos valores como entradas. El último módulo describe el contador con despeje sincrónico.

Ejemplo HDL 8-4

```
//Descripción estructural del ejemplo de diseño
//Ver diagrama de bloques, figura 8-10
module Example_Structure (S,CLK,Clr,E,F,A);
   input S,CLK,Clr;
   output E,F;
   output [4:1] A;
//Crear un ejemplar del circuito de control
   control ctl (S,A[3],A[4],CLK,Clr,T2,T1,Clear);
//Crear un ejemplar de los flip-flops E y F
   E_F EF (T1,T2,Clear,CLK,A[3],E,F);
//Crear un ejemplar del contador
   counter ctr (T1,Clear,CLK,A);
endmodule

//Circuito de control (figura 8-12)
module control (Start,A3,A4,CLK,Clr,T2,T1,Clear);
   input Start,A3,A4,CLK,Clr;
   output T2,T1,Clear;
   wire G1,G0,DG1,DG0;
//Circuito combinacional
   assign  DG1 = A3 & A4 & T1,
           DG0 = (Start & ~G0) | T1,
           T2  = G1,
           T1  = G0 & ~G1,
           Clear = Start & ~G0;
//Crear un ejemplar del flip-flop D
   DFF G1F (G1,DG1,CLK,Clr),
       G0F (G0,DG0,CLK,Clr);
endmodule

//Flip-flop D
module DFF (Q,D,CLK,Clr);
   input D,CLK,Clr;
   output Q;
   reg Q;
   always @ (posedge CLK or negedge Clr)
     if (~Clr) Q = 1'b0;
     else Q = D;
endmodule

//Flip-flops E y F
module E_F (T1,T2,Clear,CLK,A3,E,F);
   input T1,T2,Clear,CLK,A3;
   output E,F;
   wire E,F,JE,KE,JF,KF;
```

(continúa)

```
//Circuito combinacional
   assign JE = T1 & A3,
          KE = T1 & ~A3,
          JF = T2,
          KF = Clear;
//Crear un ejemplar del flip-flop JK
   JKFF EF (E,JE,KE,CLK),
        FF (F,JF,KF,CLK);
endmodule

//Flip-flop JK
module JKFF (Q,J,K,CLK);
   input J,K,CLK;
   output Q;
   reg Q;
   always @ (posedge CLK)
     case ({J,K})
       2'b00: Q = Q;
       2'b01: Q = 1'b0;
       2'b10: Q = 1'b1;
       2'b11: Q = ~Q;
     endcase
endmodule

//Contador con despeje sincrónico
module counter (Count,Clear,CLK,A);
   input Count,Clear,CLK;
   output [4:1] A;
   reg [4:1] A;
   always @ (posedge CLK)
     if (Clear) A<= 4'b0000;
     else if (Count) A <= A + 1'b1;
     else A <= A;
endmodule
```

La descripción estructural se probó con el conjunto de pruebas del ejemplo 8-3. El único cambio necesario es sustituir la creación del ejemplar de Example_RTL a Example_Structure. El resultado de la simulación para la descripción estructural es igual a la simulación de salida que se obtuvo de la descripción RTL.

8-6 MULTIPLICADOR BINARIO

En esta sección se presenta un segundo ejemplo de diseño. Se presenta un algoritmo en hardware para la multiplicación binaria, se propone la configuración de registros para su implementación y luego se muestra el diseño del camino de datos y el control con un diagrama ASM.

El sistema que examinaremos multiplica dos números binarios sin signo. En la sección 4-6 desarrollamos un algoritmo en hardware para ejecutar la multiplicación, que produjo un circuito combinacional multiplicador con muchos sumadores y compuertas AND. En esta sección, en cambio, el algoritmo en hardware producirá un multiplicador secuencial que sólo utiliza un sumador y un registro de desplazamiento.

La multiplicación de dos números binarios se efectúa con lápiz y papel efectuando sumas y desplazamientos sucesivos. La mejor forma de explicar este proceso es con un ejemplo numérico. Multipliquemos los dos números binarios 10111 y 10011:

```
 23         10111     multiplicando
 19         10011     multiplicador
---         -----
            10111
           10111
          00000
         00000
        10111
---     ---------
437     110110101     producto
```

El proceso consiste en examinar bits sucesivos del multiplicador, comenzando por el menos significativo. Si el bit del multiplicador es 1, el multiplicando se copia abajo; si no, se copian ceros. Los números copiados en líneas sucesivas se desplazan una posición a la izquierda respecto al número anterior. Por último, los números se suman y su suma forma el producto. El producto obtenido de la multiplicación de dos números binarios de *n* bits cada uno puede tener hasta 2*n* bits.

Cuando el procedimiento de multiplicación se implementa con hardware digital, conviene modificar un poco el proceso. En primer lugar, en vez de incluir circuitos digitales para almacenar y sumar simultáneamente tantos números binarios como unos haya en el multiplicador, resulta menos costoso incluir circuitos para sumar sólo dos números binarios y acumular sucesivamente los productos parciales en un registro. En segundo lugar, en lugar de desplazar el multiplicando a la izquierda, el producto parcial que se está formando se desplaza a la derecha. Esto deja al producto parcial y al multiplicando en las posiciones relativas requeridas. En tercer lugar, si el bit correspondiente del multiplicador es 0, no es necesario sumar ceros al producto parcial, ya que esto no alterará su valor.

Configuración de registros

En la figura 8-13 se observa el diagrama de bloques del multiplicador binario. El multiplicando se guarda en el registro *B*, el multiplicador se guarda en el registro *Q*, y el producto parcial se forma en el registro *A* y se guarda en *A* y *Q*. Un sumador paralelo suma el contenido de los registros *B* y *A*. El flip-flop *C* almacena el acarreo después de la suma. El contador *P* se ajusta de modo que inicialmente contenga un número binario igual al número de bits del multiplicador. Este contador se decrementa después de formarse cada producto parcial. Cuando el contenido del contador llega a cero, se forma el producto en el registro doble *AQ* y el proceso termina. La lógica de control permanece en un estado inicial hasta que la señal de inicio *S* cambia a 1. Entonces, el sistema efectúa la multiplicación. La suma de *A* y *B* forma los *n* bits más significativos del producto parcial, que se transfiere a *A*. El acarreo de salida de la suma (C_{salida}), sea 0 o 1, se transfiere a *C*. Tanto el producto parcial que está en *A* como el multiplicador que

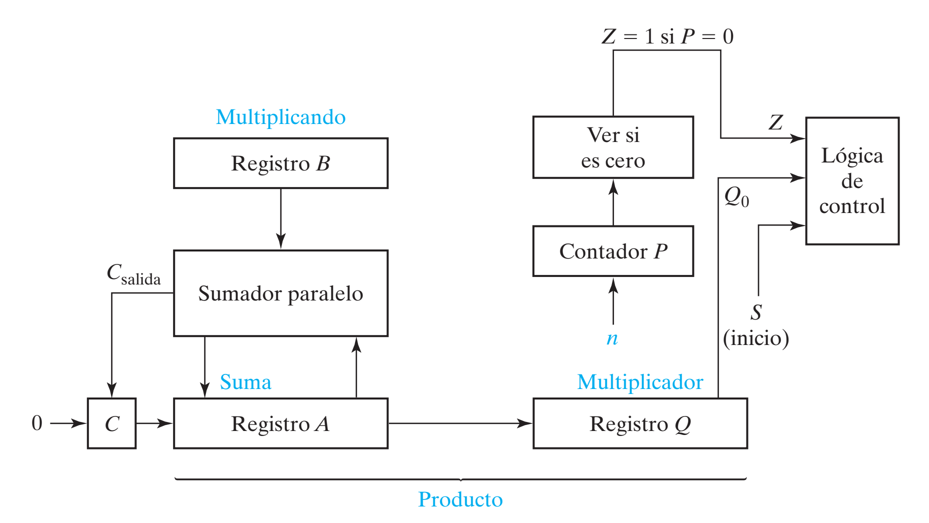


FIGURA 8-13
Diagrama de bloques del multiplicador binario

está en Q se desplazan a la derecha. El bit menos significativo de A se desplaza a la posición más significativa de Q; el acarreo de C se desplaza a la posición más significativa de A; y se desplaza 0 a C. Después de la operación de desplazamiento a la derecha, un bit del producto parcial se transfiere a Q mientras los bits del multiplicador que están en Q se desplazan una posición a la derecha. De este modo, el bit menos significativo del registro Q, Q_0, contiene el bit del multiplicador que es preciso examinar a continuación. La lógica de control determina si debe sumar o no, con base en este bit de entrada. La lógica de control también recibe la señal Z de un circuito que verifica si el contador P ha llegado a cero o no. Q_0 y Z son entradas de estado de la unidad de control. La entrada de inicio S es una entrada de control externa. Las salidas de la lógica de control activan las operaciones requeridas en los registros.

Diagrama ASM

El diagrama ASM para el multiplicador binario se muestra en la figura 8-14. En un principio, el multiplicando está en B y el multiplicador está en Q. En tanto el circuito esté en el estado inicial y $S = 0$, no se efectuará ninguna acción y el sistema permanecerá en el estado inicial T_0. El proceso de multiplicación inicia cuando $S = 1$ y el control pasa al estado T_1. El registro A y el flip-flop de acarreo C se ponen en ceros y el contador sucesivo P se ajusta a un número binario n, igual al número de bits del multiplicador. Luego el sistema pasa al estado T_2. Se examina el bit del multiplicador que está en Q_0 y, si es 1, el multiplicando que está en B se suma al producto parcial que está en A. El acarreo de la suma se transfiere a C. Si $Q_0 = 0$, no se modifican ni el producto parcial en A ni C. El contador P se decrementa en 1, sea cual sea el valor de Q_0. En ambos casos, el siguiente estado es T_3. Los registros C, A y Q se combinan en un registro compuesto CAQ, y su contenido se desplaza una posición a la derecha para obtener un nuevo producto parcial. Esta operación de desplazamiento a la derecha (*shift right*, en inglés) se indica en el diagrama de forma compacta con el enunciado

$$\text{shr } CAQ, C \leftarrow 0$$

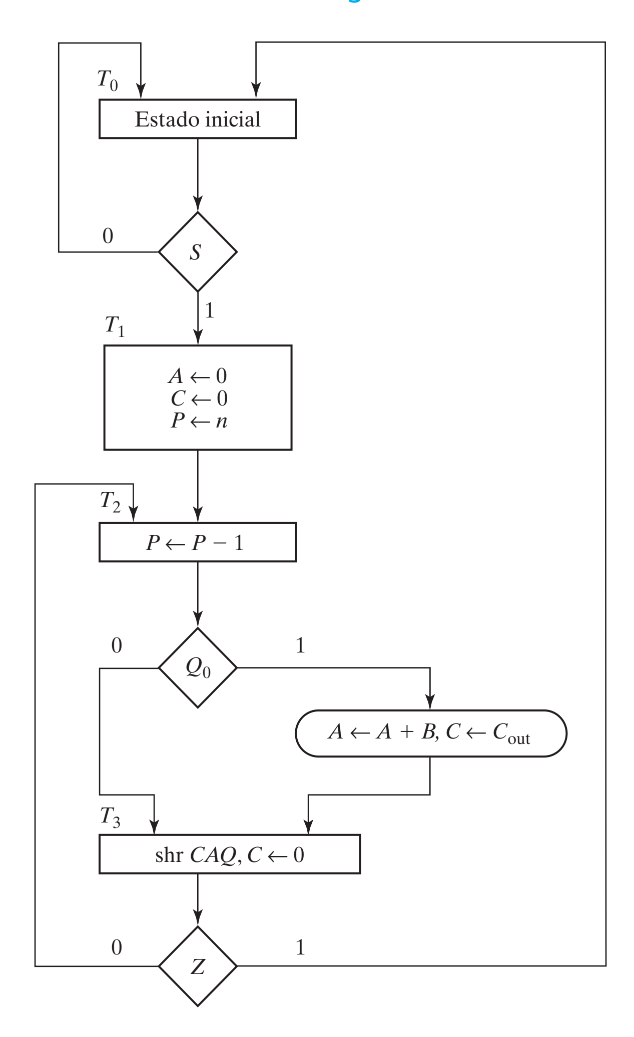


FIGURA 8-14
Diagrama ASM para el multiplicador binario

Utilizando símbolos de registros individuales, la operación se describe con las operaciones de registros siguientes:

$$A \leftarrow \text{shr } A,\ A_{n-1} \leftarrow C$$
$$Q \leftarrow \text{shr } Q,\ Q_{n-1} \leftarrow A_0$$
$$C \leftarrow 0$$

Ambos registros, A y Q, se desplazan a la derecha. El bit de la extrema izquierda de A, designado por A_{n-1}, recibe el acarreo de C. El bit de la extrema izquierda de Q, Q_{n-1}, recibe el bit de la posición extrema derecha de A, en A_0; y C se restablece a 0. En esencia, se trata de un desplazamiento largo del registro compuesto CAQ, insertando 0 en la entrada en serie, que está en C.

Tabla 8-4
Ejemplo numérico para el multiplicador binario

Multiplicando B = 10111

	C	A	Q	P
Multiplicador en Q	0	00000	10011	101
$Q_0 = 1$; sumar B		10111		
Primer producto parcial	0	10111		100
Desplazar CAQ a la derecha	0	01011	11001	
$Q_0 = 1$; sumar B		10111		
Segundo producto parcial	1	00010		011
Desplazar CAQ a la derecha	0	10001	01100	
$Q_0 = 0$; desplazar CAQ a la derecha	0	01000	10110	010
$Q_0 = 0$; desplazar CAQ a la derecha	0	00100	01011	001
$Q_0 = 1$; sumar B		10111		
Quinto producto parcial	0	11011		
Desplazar CAQ a la derecha	0	01101	10101	000
Producto final en AQ = 0110110101				

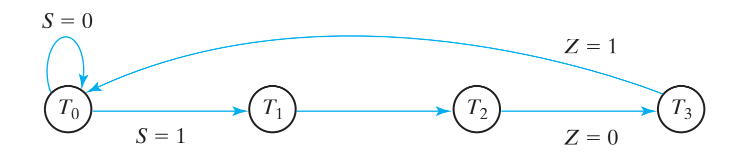
El valor del contador P se examina después de la formación de cada producto parcial. Si el contenido de P no es cero, el bit de estado Z es 0 y el proceso se repite para formar un nuevo producto parcial. El proceso se detiene cuando el contador P llega a 0 y la entrada de control Z es 1. El producto parcial formado en A se desplaza a Q bit por bit y finalmente reemplaza al multiplicador. El producto final queda en A y Q, con los bits más significativos del producto en A, y los menos significativos, en Q.

En la tabla 8-4 se repite el ejemplo numérico anterior para aclarar el proceso de multiplicación. El procedimiento sigue los pasos delineados en el diagrama ASM.

El tipo de registros que se requieren para el subsistema procesador de datos se deduce de las operaciones de registros enumeradas en el diagrama ASM. El registro A es un registro de desplazamiento con carga paralela para aceptar la suma del sumador, y requiere capacidad de despeje sincrónico para restablecerse a 0. El registro Q es un registro de desplazamiento. El contador P es un contador binario regresivo que puede cargar en paralelo una constante binaria. El flip-flop C debe diseñarse de modo que acepte el acarreo de entrada y se pueda despejar sincrónicamente. Los registros B y Q requieren capacidad de carga en paralelo para recibir el multiplicando y el multiplicador antes de que se inicie el proceso de multiplicación.

8-7 LÓGICA DE CONTROL

El diseño de un sistema digital se divide en dos partes: el diseño de las transferencias de registros en el camino de datos y el diseño de la lógica de control. El diseño de la lógica de control es un problema de diseño de circuitos secuenciales. Como tal, podría ser conveniente formular el diagrama de estados del control secuencial. Los diagramas ASM son similares a los diagramas de estados. Los rectángulos que denotan cuadros de estado son los estados del circuito secuen-



a) Diagrama de estados

T_0: Estado inicial

T_1: $A \leftarrow 0, C \leftarrow 0, P \leftarrow n$

T_2: $P \leftarrow P - 1$

if $(Q_0) = 1$ then $(A \leftarrow A + B, C \leftarrow C_{out})$

T_3: desplazar a la derecha CAQ, $C \leftarrow 0$

b) Operaciones de transferencia de registros

FIGURA 8-15
Especificaciones de control para el multiplicador binario

cial. Los rombos que denotan cuadros de decisión determinan las condiciones para la transición al siguiente estado del diagrama de estados. En la figura 8-15 se presenta, como ejemplo, el diagrama de estados de control para el multiplicador binario desarrollado en la sección anterior. La información para el diagrama se tomó directamente del diagrama ASM de la figura 8-14. Los cuatro estados, T_0 a T_3, se toman de los cuadros de estado rectangulares. Las entradas S y Z se toman de los cuadros de decisión rómbicos. Las operaciones de transferencia de registros para cada uno de los cuatro estados se dan abajo del diagrama de estados. Se tomaron de los cuadros de estado y condicionales correspondientes del diagrama ASM.

Hay dos tareas distintas que debemos efectuar al implementar la lógica de control: establecer la sucesión requerida de estados y generar señales que controlen las operaciones de registros. La sucesión de estados se especifica en el diagrama de estados. Las señales para controlar las operaciones en los registros se especifican en los enunciados de transferencia de registros. En el caso del multiplicador, dichas señales son T_1 (para despejar A y C y cargar un número en P), T_2 (para decrementar P), T_3 (para desplazar el registro CAQ) y Q_0 para determinar si B se suma a A o no. El diagrama de bloques del control se reproduce en la figura 8-16. Las entradas para el control secuencial son S y Z, y las salidas, T_0, T_1, T_2, T_3, como especifica el diagrama de estados. La compuerta AND que genera la señal $L = T_2Q_0$ se necesita para cargar la suma en el registro A si $Q_0 = 1$ estando en el estado T_2.

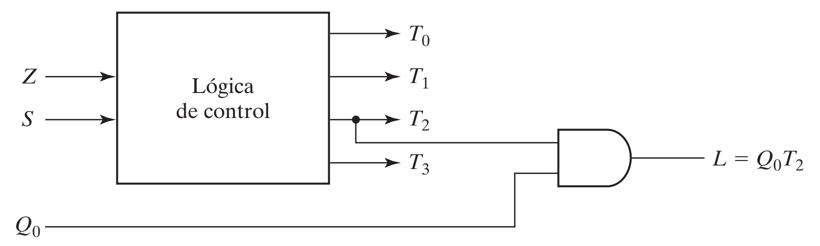


FIGURA 8-16
Diagrama de bloques del control

Un paso importante del diseño es la asignación de valores binarios codificados a los estados. La asignación más sencilla es la sucesión binaria directa que aparece en la tabla 8-5. Otra asignación similar es el código Gray, en el que sólo un bit cambia al pasar de un número al siguiente. Una asignación de estados que se usa mucho en diseños de control es la asignación de *un solo uno*. Esta asignación utiliza tantos bits como estados hay en el circuito. En cualquier momento dado, sólo un bit es 1, mientras todos los demás se mantienen en 0. Este tipo de asignación usa un flip-flop para cada estado.

Tabla 8-5
Asignación de estados para el control

Estado	Binario	Código Gray	Un solo uno
T_0	00	00	0001
T_1	01	01	0010
T_2	10	11	0100
T_3	11	10	1000

Puesto que el control es un circuito secuencial, podemos diseñarlo siguiendo el procedimiento de lógica secuencial delineado en el capítulo 5. Sin embargo, en la mayoría de los casos este método no resulta práctico debido al gran número de estados y entradas que puede tener un circuito de control típico. Por ello, es necesario utilizar métodos especializados para diseñar lógica de control, los cuales podrían considerarse variaciones del método de lógica secuencial clásico. A continuación se presentan dos de esos procedimientos de diseño. Uno emplea un registro de secuencia y un decodificador; el otro usa un flip-flop por estado.

Registro de secuencia y decodificador

El método de registro de secuencia y decodificador, como su nombre implica, utiliza un registro para los estados de control y un decodificador para generar una salida correspondiente a cada uno de los estados. Un registro con n flip-flops puede tener hasta 2^n estados, y un decodificador de n a 2^n líneas tiene hasta 2^n salidas. Un registro de secuencia de n bits es, en esencia, un circuito con n flip-flops y las compuertas asociadas que efectúan su transición de estado.

El diagrama de estados de control para el multiplicador binario tiene cuatro estados y dos entradas. Para implementarlo con un registro de secuencia y un decodificador, necesitamos

Tabla 8-6
Tabla de estados para el circuito de control

Estado actual		Entrada		Siguiente estado		Salidas			
G_1	G_0	S	Z	G_1	G_0	T_0	T_1	T_2	T_3
0	0	0	X	0	0	1	0	0	0
0	0	1	X	0	1	1	0	0	0
0	1	X	X	1	0	0	1	0	0
1	0	X	X	1	1	0	0	1	0
1	1	X	0	1	0	0	0	0	1
1	1	X	1	0	0	0	0	0	1

dos flip-flops para el registro y un decodificador de 2 a 4 líneas. Aunque se trata de un ejemplo sencillo, el procedimiento que bosquejaremos se aplica también en situaciones más complejas.

La tabla de estados para el control secuencial se muestra en la tabla 8-6. Se dedujo directamente del diagrama de estados de la figura 8-15a). Designamos los dos flip-flops como G_1 y G_0 y asignamos los estados binarios 00, 01, 10 y 11 a T_0, T_1, T_2 y T_3, respectivamente. Las columnas de entrada tienen indicadores de indiferencia en los casos en que la variable de entrada no se utiliza para determinar el siguiente estado. Las salidas del circuito de control se denotan con los nombres de los estados. La variable de salida que es 1 en un momento dado se determina a partir del valor binario equivalente del estado actual. Así pues, cuando el estado actual es $G_1G_0 = 00$, la salida T_0 debe ser 1, mientras que las demás salidas permanecen en 0. Puesto que las salidas sólo son función del estado actual, se pueden generar con un circuito decodificador de dos entradas, G_1 y G_0, y cuatro salidas, T_0 a T_3.

El circuito secuencial se puede diseñar a partir de la tabla de estados utilizando el procedimiento clásico que se presentó en el capítulo 5. Este ejemplo tiene pocos estados y entradas, así que pudimos usar mapas para simplificar las funciones booleanas. En casi todas las aplicaciones de lógica de control, el número de estados y entradas es mucho mayor. La aplicación del método clásico requiere una cantidad excesiva de trabajo para obtener las ecuaciones de entrada de los flip-flops. El diseño se simplifica si tomamos en consideración el hecho de que las salidas del decodificador se pueden utilizar en el diseño. En lugar de usar salidas de flip-flop como condiciones de estado actual, bien podríamos usar las salidas del decodificador para suministrar las condiciones de estado actual del circuito secuencial. Además, en lugar de usar mapas para simplificar las ecuaciones de los flip-flops, podríamos obtenerlas directamente por inspección de la tabla de estados. Por ejemplo, de las condiciones de siguiente estado de la tabla de estados deducimos que el siguiente estado de G_1 es 1 si el estado actual es T_1, T_2 o T_3, a condición de que $Z = 0$. Estas condiciones se especifican con la ecuación:

$$D_{G1} = T_1 + T_2 + T_3Z'$$

donde D_{G1} es la entrada D del flip-flop G_1. Asimismo, la entrada D de G_0 es

$$D_{G0} = T_0S + T_2$$

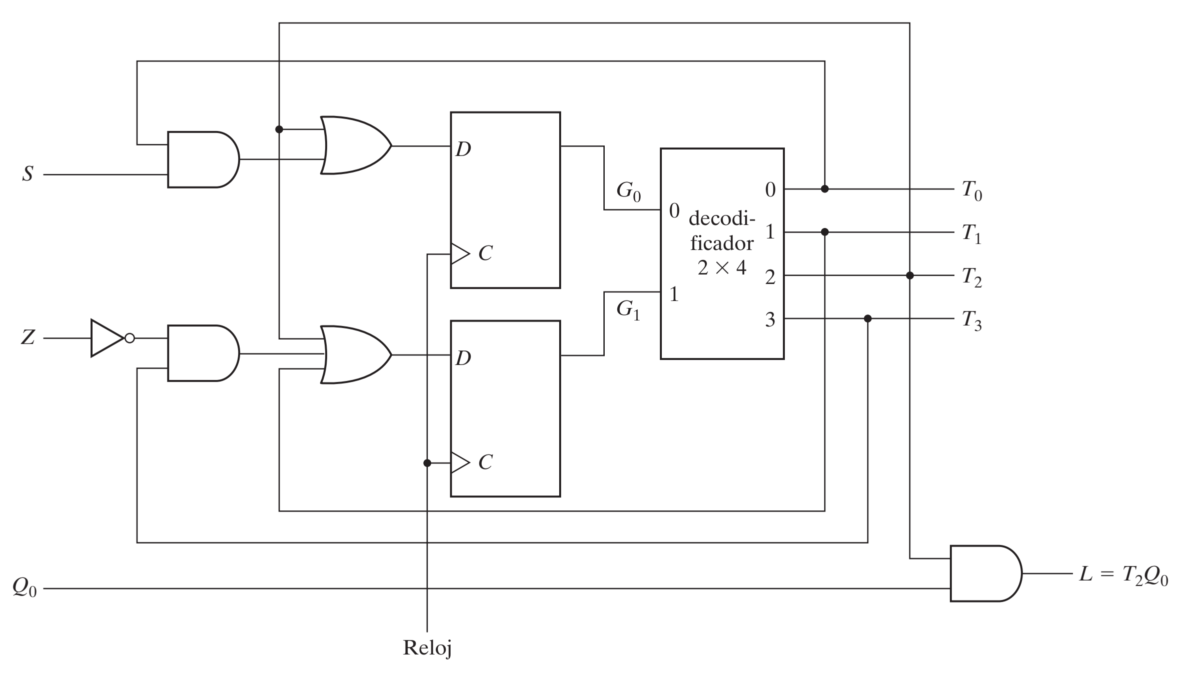


FIGURA 8-17
Diagrama lógico del control para el multiplicador binario empleando un registro de secuencia y un decodificador

Al deducir ecuaciones de entrada por inspección de la tabla de estados, no podemos tener la certeza de que las ecuaciones booleanas se han simplificado de manera óptima. Por ello, es aconsejable analizar el circuito para verificar que las ecuaciones deducidas realmente produzcan las transiciones de estado deseadas.

El diagrama lógico del circuito de control se ha dibujado en la figura 8-17. Consiste en un registro con dos flip-flops, G_1 y G_0, y un decodificador 2×4. Las salidas del decodificador se utilizan para generar las entradas de la lógica de siguiente estado, así como las salidas de control. Las salidas del controlador deben conectarse a la trayectoria de datos para activar las operaciones de registros requeridas.

Un flip-flop por estado

Otro posible método para diseñar la lógica de control es utilizar la asignación de un solo uno que produce un circuito secuencial con un flip-flop por estado. Sólo uno de los flip-flops contiene un 1 en cualquier momento dado; todos los demás se restablecen a 0. El 1 se propaga de un flip-flop a otro bajo el control de la lógica de decisión. En una configuración así, cada flip-flop representa un estado que sólo está presente cuando el bit de control se transfiere a él.

Este método utiliza el número máximo de flip-flops para el circuito secuencial. Por ejemplo, un circuito secuencial con 12 estados requiere por lo menos cuatro flip-flops. Si se utiliza el método de un flip-flop por estado, el circuito requerirá 12 flip-flops, uno para cada estado. A primera vista, podría parecer que este método aumenta el costo del sistema porque se usan

más flip-flops. No obstante, el método ofrece ciertas ventajas que tal vez no sean obvias. Una es la sencillez con que puede diseñarse la lógica con sólo inspeccionar el diagrama ASM o el diagrama de estados. No se necesitan tablas de estados ni de excitación si se usan flip-flops tipo D. El diseño requiere menos esfuerzo, y la sencillez operativa es mayor, además de que podría reducirse el número total de compuertas, pues no se necesita un decodificador.

Ilustraremos el procedimiento de diseño produciendo el circuito de control especificado por el diagrama de estados de la figura 8-15a). Puesto que hay cuatro estados en ese diagrama, escogemos cuatro flip-flops D y rotulamos sus salidas T_0, T_1, T_2 y T_3. Las ecuaciones de entrada para poner en 1 cada flip-flop se deducen del estado actual y de las condiciones de entrada que están sobre las flechas correspondientes que llegan al estado. Por ejemplo, el flip-flop T_0 se pone en 1 con el siguiente borde de reloj si el circuito está en el estado T_3 y la entrada Z es 1, o si el circuito está en el estado T_0 y S es 0. Estas condiciones se especifican con la ecuación de entrada:

$$D_{T0} = T_0 S' + T_3 Z$$

donde D_{T0} denota la entrada D del flip-flop T_0. De hecho, la condición para poner en 1 un flip-flop se obtiene directamente del diagrama de estados, efectuando un AND con la condición especificada en las flechas que llegan al estado correspondiente y el estado anterior del flip-flop. Si llegan dos o más flechas a un estado, se deberá hacer el OR de todas las condiciones. Al aplicar este procedimiento a los otros tres flip-flops, se obtienen las ecuaciones de entrada:

$$D_{T1} = T_0 S$$
$$D_{T2} = T_1 + T_3 Z'$$
$$D_{T3} = T_2$$

El diagrama lógico del controlador aparece en la figura 8-18. Consta de cuatro flip-flops D, T_0 a T_3, y las compuertas asociadas especificadas por las ecuaciones de entrada. En un principio, el flip-flop T_0 debe ponerse en 1 y todos los demás se deben poner en 0 para habilitar el flip-flop que representa el estado inicial. Esto se logra con un preestablecimiento asincrónico en el flip-flop T_0 y un despeje asincrónico en los otros flip-flops. Una vez iniciado, el controlador de un flip-flop por estado pasará de un estado al siguiente en la forma correcta. Sólo un flip-flop se encenderá con cada borde de reloj; todos los demás se apagarán porque sus entradas D son 0.

8-8 DESCRIPCIÓN DEL MULTIPLICADOR BINARIO EN HDL

El ejemplo HDL 8-5 es un segundo ejemplo de descripción HDL de un diseño RTL, y corresponde al multiplicador binario diseñado en la sección 8-6. La descripción se divide en cinco partes, cada una de las cuales va precedida por un comentario que la explica. La primera parte enumera todas las entradas y salidas especificadas en el diagrama de bloques de la figura 8-13. Las entradas de datos a B y Q y las salidas de datos de A y Q son vectores de cinco bits. Escogimos cinco bits para este ejemplo a fin de poder comparar el resultado de la multiplicación con el ejemplo numérico de la tabla 8-4. La segunda parte declara todos los registros, incluidos el registro de control y la codificación de los cuatro estados. La tercera parte especifica un circuito combinacional con un enunciado **assign.** Hay dos circuitos combinacionales en el

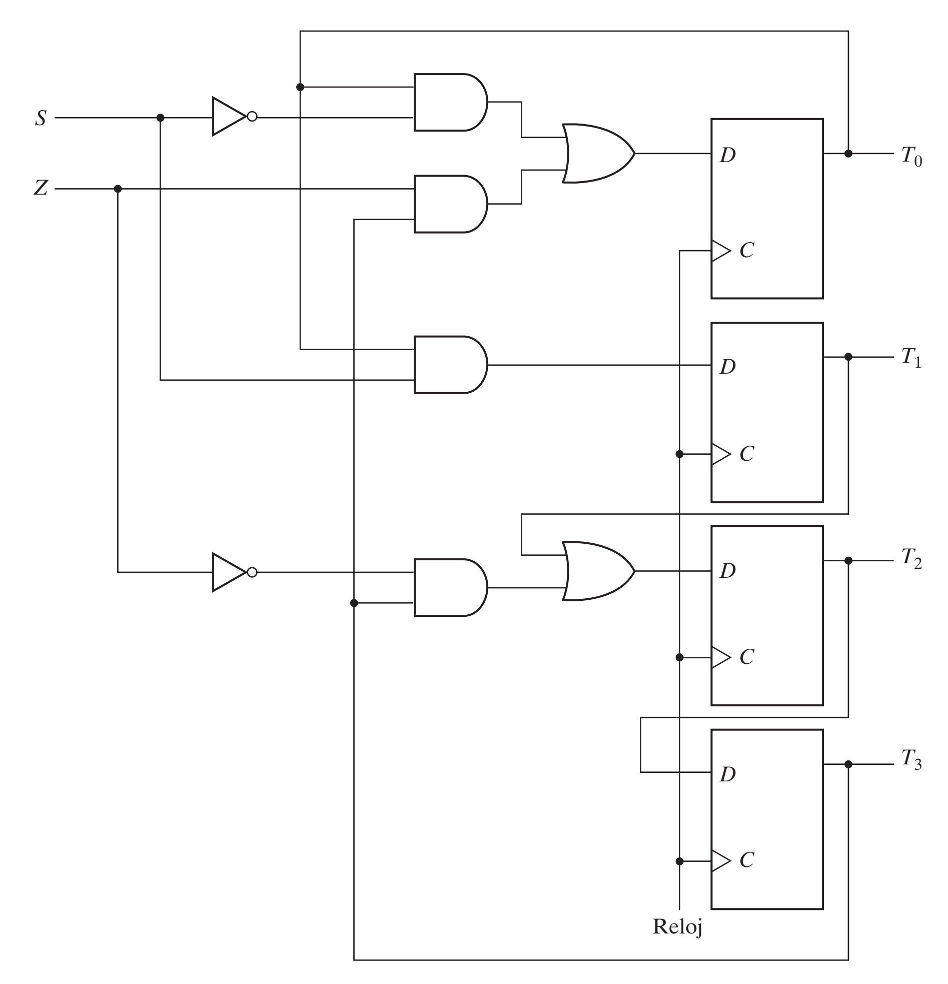


FIGURA 8-18
Controlador de un flip-flop por estado

multiplicador. Uno examina la salida del contador para detectar una salida de cero; el otro es un sumador paralelo. El sumador se especifica después como parte de la operación de transferencia de registros que suma el multiplicando al producto parcial. La detección de cero se implementa con una compuerta NOR de cinco entradas. Esto se especifica en el enunciado **assign** con un operador de reducción NOR.

La cuarta parte describe la transición de estados para el control y sigue el diagrama de estados de la figura 8-15a). La última parte es una descripción de las operaciones de transferencia de registros para la trayectoria de datos. Ésta sigue las operaciones que se presentan en el diagrama ASM de la figura 8-15b). Las entradas de datos no se dan en el diagrama ASM, pero se necesitan aquí para verificar el funcionamiento del circuito. El multiplicando se transfiere al registro *B* durante el estado inicial *T*0. El multiplicador se transfiere durante el estado *T*1. En *T*2, el multiplicando se suma al producto parcial si $Q[0] = 1$. En *T*3, el producto parcial se desplaza a la derecha.

Ejemplo HDL 8-5

```
//Descripción RTL del multiplicador binario
//correspondiente al diagrama de bloques de la figura 8-13 y al diagrama
//ASM de la figura 8-14
//n=5 para comparar con tabla 8-4
module mltp(S,CLK,Clr,Binput,Qinput,C,A,Q,P);
   input S,CLK,Clr;
   input [4:0] Binput,Qinput;          //Entradas de datos
   output C;
   output [4:0] A,Q;
   output [2:0] P;
//Registros del sistema
   reg C;
   reg [4:0] A,Q,B;
   reg [2:0] P;
   reg [1:0] pstate, nstate;           //registro de control
   parameter T0=2'b00, T1=2'b01, T2=2'b10, T3=2'b11;
//Circuito combinacional
   wire Z;
   assign Z = ~|P;                     //Ver si es cero
//Transición de estado para control
//Ver diagrama de estados figura 8-15a)
   always @(negedge CLK or negedge Clr)
     if (~Clr) pstate = T0;
     else pstate <= nstate;
   always @(S or Z or pstate)
     case (pstate)
       T0: if (S) nstate = T1; else nstate = T0;
       T1: nstate = T2;
       T2: nstate = T3;
       T3: if (Z) nstate = T0;
           else   nstate = T2;
     endcase
//Operaciones de transferencia de registros
//Ver operación de registros figura 8-15b)
   always @(negedge CLK)
     case (pstate)
       T0: B <= Binput;                //Introducir multiplicando
       T1: begin
             A <= 5'b00000;
             C <= 1'b0;
             P <= 3'b101;             //Iniciar contador en n=5
             Q <= Qinput;              //Introducir multiplicador
           end
       T2: begin
             P <= P - 3'b001;          //Decrementar contador
             if (Q[0])
             {C,A} <= A + B;           //Sumar multiplicando
           end
       T3: begin
             C <= 1'b0;                //Despejar C
             A <= {C,A[4:1]};          //Desplazar A a la derecha
             Q <= {A[0],Q[4:1]};       //Desplazar Q a la derecha
           end
     endcase
endmodule
```

Prueba del multiplicador

El ejemplo HDL 8-6 constituye un conjunto de pruebas para probar el multiplicador. Las entradas son el multiplicando y el multiplicador que están en *B* y en *Q*. La salida es el producto que queda en *A* y *Q*. También verificamos los contadores del acarreo *C* y el contador *P*. Las entradas binarias de cinco bits son las mismas que se utilizaron en la tabla 8-4. El segundo enunciado **initial** genera 13 ciclos de reloj de 10 unidades de tiempo cada uno. Una inspección de las transiciones de estados revela que los estados $T2$ y $T3$ se suceden cíclicamente cinco veces (una vez por cada producto parcial). Esto requiere 10 ciclos de reloj. Se necesitan otros tres ciclos de reloj para $T0$, $T1$ y el regreso a $T0$ cuando $Z = 1$.

El resultado del cálculo se exhibe con la tarea del sistema **$strobe**. Esta tarea es similar a las tareas **$display** y **$monitor** que se explicaron en la sección 4-11. La tarea **$strobe** ofrece un mecanismo de sincronización que garantiza que los datos se exhibirán únicamente después de haberse ejecutado todas las asignaciones de un incremento de tiempo dado. Esto es muy útil en los circuitos secuenciales sincrónicos en los que el incremento de tiempo inicia en un borde de reloj. El uso de **$strobe** después del enunciado **always @ negedge** CLK garantiza que se exhibirán los valores de las señales tal como existen después de una transición negativa del reloj.

El módulo test_mltp del ejemplo HDL 8-6 crea un ejemplar del módulo mltp del ejemplo HDL 8-5. Ambos módulos deben incluirse al simular el multiplicador con un simulador Verilog HDL. El resultado de esta simulación exhibe una bitácora de simulación con números idénticos a los de la tabla 8-4.

Descripción del comportamiento del multiplicador

El multiplicador binario se describe en el nivel de comportamiento, como se indica en el ejemplo HDL 8-7. Las dos entradas se declaran como vectores de ocho bits, y la salida, como uno de 16 bits. No hay implícito un hardware específico, como en las descripciones RTL o estructural. El funcionamiento se especifica con un solo enunciado de multiplicación. En este caso, la descripción podría sintetizarse para dar un circuito combinacional multiplicador (véase la sección 4-6) si el software de síntesis así se diseñó. En general, las descripciones de comportamiento basadas en algoritmos no siempre son sintetizables.

8-9 DISEÑO CON MULTIPLEXORES

El control de registro de secuencia y decodificador consta de tres componentes: los flip-flops que contienen el valor binario del estado, el decodificador que genera las salidas de control y las compuertas que determinan las señales de siguiente estado y de salida. En la sección 4-10 demostramos que es posible implementar un circuito combinacional con multiplexores en lugar de compuertas individuales. La sustitución de las compuertas por multiplexores produce un patrón regular de tres niveles de componentes. El primer nivel consiste en multiplexores que determinan el siguiente estado del registro. El segundo nivel contiene un registro que guarda el estado actual binario. El tercer nivel tiene un decodificador que genera una salida individual para cada estado de control. Estos tres componentes son celdas estándar predefinidas en muchos circuitos integrados.

Consideremos, por ejemplo, el diagrama ASM de la figura 8-19. Consta de cuatro estados y cuatro entradas de control. Los cuadros de estado se han dejado vacíos en este caso porque sólo nos interesa la sucesión de control, que es independiente de las operaciones de registros. La asignación binaria a cada estado se indica en la esquina superior derecha de los cuadros de estado. Los cuadros de decisión especifican las transiciones de estado en función de las cuatro entradas

Ejemplo HDL 8-6

```
//Prueba del multiplicador binario
module test_mltp;
//Entradas del multiplicador
   reg S,CLK,Clr;
   reg [4:0] Binput,Qinput;
//Datos a exhibir
   wire C;
   wire [4:0] A,Q;
   wire [2:0] P;
//Crear ejemplar del multiplicador
   mltp mp(S,CLK,Clr,Binput,Qinput,C,A,Q,P);
   initial
     begin
          S=0; CLK=0; Clr=0;
      #5  S=1; Clr=1;
          Binput = 5'b10111;
          Qinput = 5'b10011;
      #15 S = 0;
     end
   initial
     begin
         repeat (26)
      #5 CLK = ~CLK;
     end
//Exhibir cálculos y comparar con tabla 8-4
   always @(negedge CLK)
     $strobe ("C=%b  A=%b  Q=%b  P=%b  time=%0d",C,A,Q,P,$time);
endmodule


Simulation log:

C=x  A=xxxxx  Q=xxxxx  P=xxx  time=10
C=0  A=00000  Q=10011  P=101  time=20
C=0  A=10111  Q=10011  P=100  time=30
C=0  A=01011  Q=11001  P=100  time=40
C=1  A=00010  Q=11001  P=011  time=50
C=0  A=10001  Q=01100  P=011  time=60
C=0  A=10001  Q=01100  P=010  time=70
C=0  A=01000  Q=10110  P=010  time=80
C=0  A=01000  Q=10110  P=001  time=90
C=0  A=00100  Q=01011  P=001  time=100
C=0  A=11011  Q=01011  P=000  time=110
C=0  A=01101  Q=10101  P=000  time=120
C=0  A=01101  Q=10101  P=000  time=130
```

Ejemplo HDL 8-7

```
//Descripción de comportamiento del multiplicador (n = 8)
module Mult (A,B,Q);
   input [7:0] B,Q;
   output [15:0] A;
   reg [15:0] A;
   always @ (B or Q)
     A = B * Q;
endmodule
```

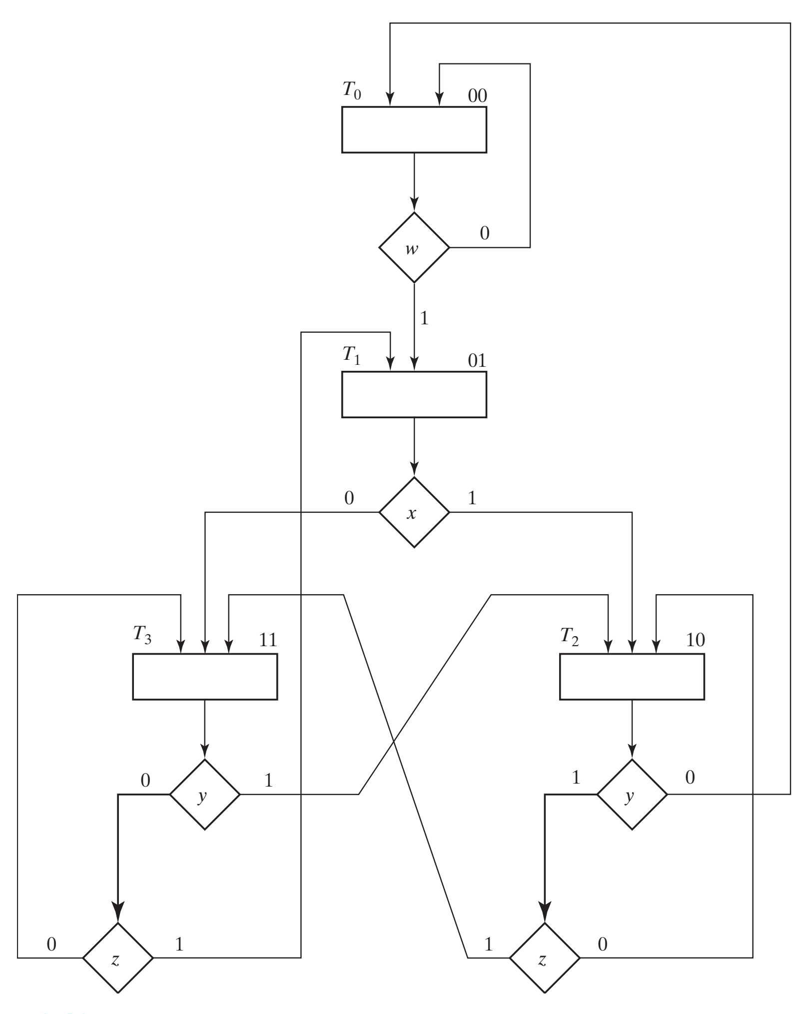


FIGURA 8-19
Ejemplo de diagrama ASM con cuatro entradas de control

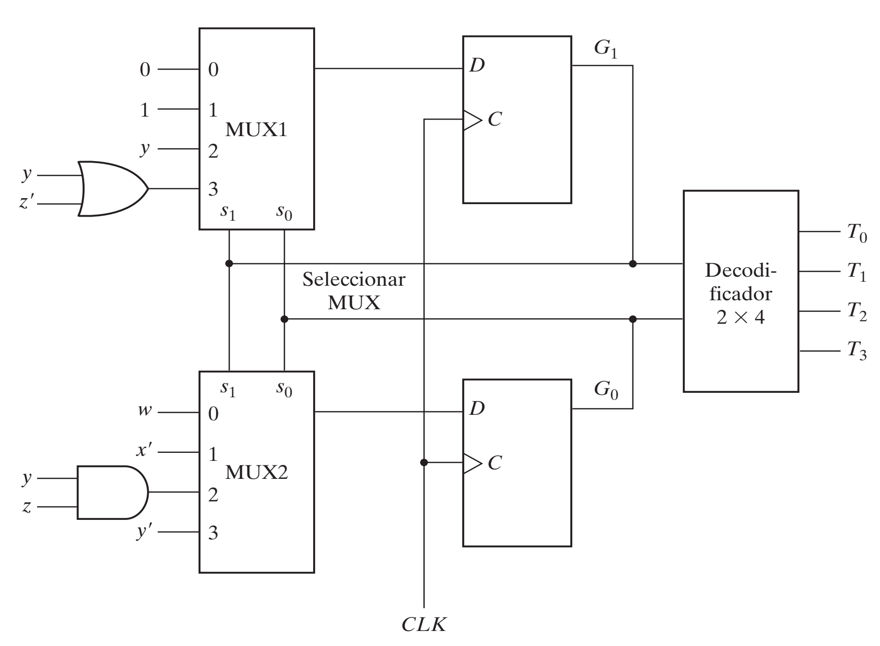


FIGURA 8-20
Implementación del control con multiplexores

de control: w, x, y y z. La implementación del control en tres niveles se ilustra en la figura 8-20. Consiste en dos multiplexores, MUX1 y MUX2; un registro con dos flip-flops, G_1 y G_0; y un decodificador con cuatro salidas. Las salidas del registro se aplican a las entradas del decodificador y también a las entradas de selección de los multiplexores. Así, el estado actual del registro sirve para seleccionar una de las entradas de cada multiplexor. Después, las salidas de los multiplexores se aplican a las entradas D de G_1 y G_0. El propósito de cada multiplexor es producir una entrada para su flip-flop correspondiente, igual al valor binario del siguiente estado. Las entradas de los multiplexores se determinan a partir de los cuadros de decisión y las transiciones de estado dadas en el diagrama ASM. Por ejemplo, el estado 00 permanece en 00 o pasa a 01, dependiendo del valor de la entrada w. Puesto que el siguiente estado de G_1 es 0 en ambos casos, colocamos una señal equivalente a 0 lógico en la entrada 0 de MUX1. El siguiente estado de G_0 es 0 si $w = 0$, y 1 si $w = 1$. Puesto que el siguiente estado de G_0 es igual a w, aplicamos la entrada de control w a la entrada 0 de MUX2. Lo que esto significa es que, cuando las entradas de selección de los multiplexores son iguales al estado actual 00, las salidas de los multiplexores generan el valor binario que se transfiere al registro durante el siguiente pulso de reloj.

Para facilitar la evaluación de las entradas de los multiplexores, preparamos una tabla que especifica las condiciones de entrada para cada posible transición en el diagrama ASM. La tabla 8-7 proporciona esta información para el diagrama ASM de la figura 8-19. Hay dos transiciones desde el estado actual 00 o 01 y tres transiciones desde el estado actual 10 u 11. Éstas se han separado con líneas horizontales en la tabla. Las condiciones de entrada que se dan en la tabla se obtienen de los cuadros de decisión del diagrama ASM. Por ejemplo, en la figura 8-19 vemos que el estado 01 pasa al estado 10 si $x = 1$, o al estado 11 si $x = 0$. En la tabla, marcamos estas condiciones de entrada como x y x', respectivamente. Las dos columnas bajo el título de "Entradas" (de multiplexor) en la tabla especifican los valores de entrada que deben aplicarse a MUX1 y MUX2. La entrada de multiplexor para cada estado actual se determina a par-

Tabla 8-7
Condiciones de entrada de los multiplexores

Estado actual		Siguiente estado		Condiciones de entrada	Entradas	
G_1	G_0	G_1	G_0		MUX1	MUX2
0	0	0	0	w'		
0	0	0	1	w	0	w
0	1	1	0	x		
0	1	1	1	x'	1	x'
1	0	0	0	y'		
1	0	1	0	yz'	$yz' + yz = y$	yz
1	0	1	1	yz		
1	1	0	1	$y'z$		
1	1	1	0	y		
1	1	1	1	$y'z'$	$y + y'z' = y + z'$	$y'z + y'z' = y'$

tir de las condiciones de entrada cuando el siguiente estado del flip-flop es 1. Así pues, después del estado actual 01, el siguiente estado de G_1 siempre es igual a 1 y el siguiente estado de G_0 es igual al complemento del valor de x. Por tanto, la entrada de MUX1 se hace igual a 1, y la de MUX2, a x', cuando el estado actual del registro es 01. Como ejemplo adicional, después del estado actual 10, el siguiente estado de G_1 debe ser 1 si las condiciones de entrada son yz' o yz. Si se obtiene el OR de estos dos términos booleanos y se simplifica el resultado, se obtiene la variable binaria única y, como se indica en la tabla. El siguiente estado de G_0 es 1 si las condiciones de entrada son $yz = 11$. Si el siguiente estado de G_1 permanece en 0 después de un estado actual dado, colocamos un 0 en la entrada del multiplexor, como se muestra en el estado actual 00 para MUX1. Si el siguiente estado de G_1 siempre es 1, colocamos un 1 en la entrada del multiplexor, como se indica en el estado actual 01 para MUX1. Las demás entradas para MUX1 y MUX2 se deducen de forma similar. Las entradas de multiplexor de la tabla se usan entonces en la implementación del control de la figura 8-20. Si el siguiente estado de un flip-flop es función de dos o más variables de control, el multiplexor podría requerir una o más compuertas en su entrada. De lo contrario, la entrada del multiplexor es igual a la variable de control, o al complemento de la variable de control, o 0, o 1.

Ejemplo de diseño: contar el número de unos en un registro

Ilustraremos la implementación de control con multiplexores empleando un ejemplo de diseño. El ejemplo también ilustrará la formulación del diagrama ASM y la implementación del subsistema de trayectoria de datos.

El sistema digital a diseñar consta de dos registros, $R1$ y $R2$, y un flip-flop, E. El sistema cuenta el número de unos que hay en el número que se carga en el registro $R1$, y pone esa cuenta en el registro $R2$. Por ejemplo, si el número binario cargado en $R1$ es 10111001, el circuito contará los cinco unos de $R1$ y guardará la cuenta binaria 101 en el registro $R2$. Esto se hace desplazando cada bit del registro $R1$, uno por uno, al flip-flop E. El control examina el valor de E, y cada vez que es 1, incrementa en 1 el registro $R2$.

El control utiliza una entrada externa S para iniciar la operación y dos entradas de estado E y Z de la trayectoria de datos. E es la salida del flip-flop. Z es la salida de un circuito que determina si el registro $R1$ contiene únicamente ceros. El circuito produce una salida $Z = 1$ si $R1$ es igual a 0.

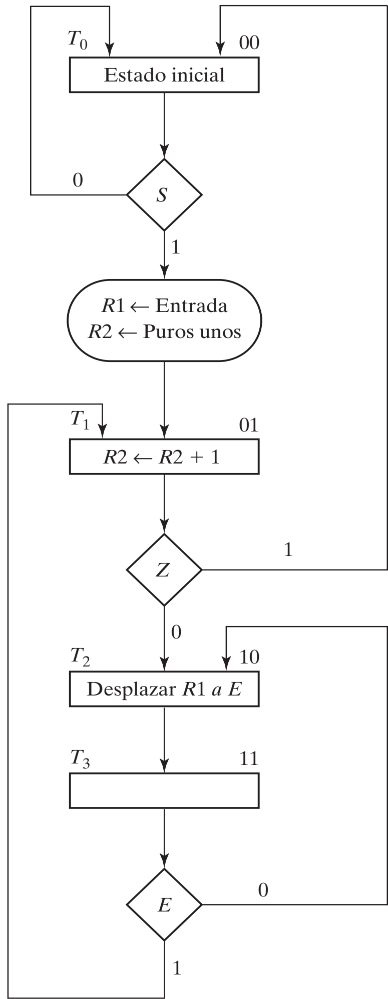


FIGURA 8-21
Diagrama ASM para el circuito contador de unos

El diagrama ASM para el circuito se aprecia en la figura 8-21. El número binario se carga en $R1$, y el registro $R2$ se llena de unos. Cuando se incrementa un número que consta de puros unos y que está guardado en un registro, el resultado es un número con puros ceros. En el estado T_1, el registro $R2$ se incrementa y se examina el contenido de $R1$. Si el contenido es cero, entonces $Z = 1$ y sabemos que no hay unos almacenados en el registro; por tanto, la operación termina con $R2$ igual a cero. Si el contenido de $R1$ no es cero, entonces $Z = 0$ y sabemos que hay por lo menos un 1 en el registro. El número almacenado en $R1$ se desplaza y su bit de extrema izquierda se transfiere a E. Esto se hace todas las veces necesarias hasta que se transfiere un 1 a E. Por cada 1 detectado en E, el registro $R2$ se incrementa y se vuelve a examinar el registro $R1$ para ver si queda algún 1. El ciclo principal se repite hasta que se cuentan todos los unos de $R1$. Note que el cuadro de estado de T_3 no tiene operaciones de registros, pero su bloque contiene el cuadro de decisión para E. Observe también que la entrada en serie al registro de desplazamiento $R1$ debe ser 0 porque no queremos introducir unos externos a $R1$ al desplazarlo.

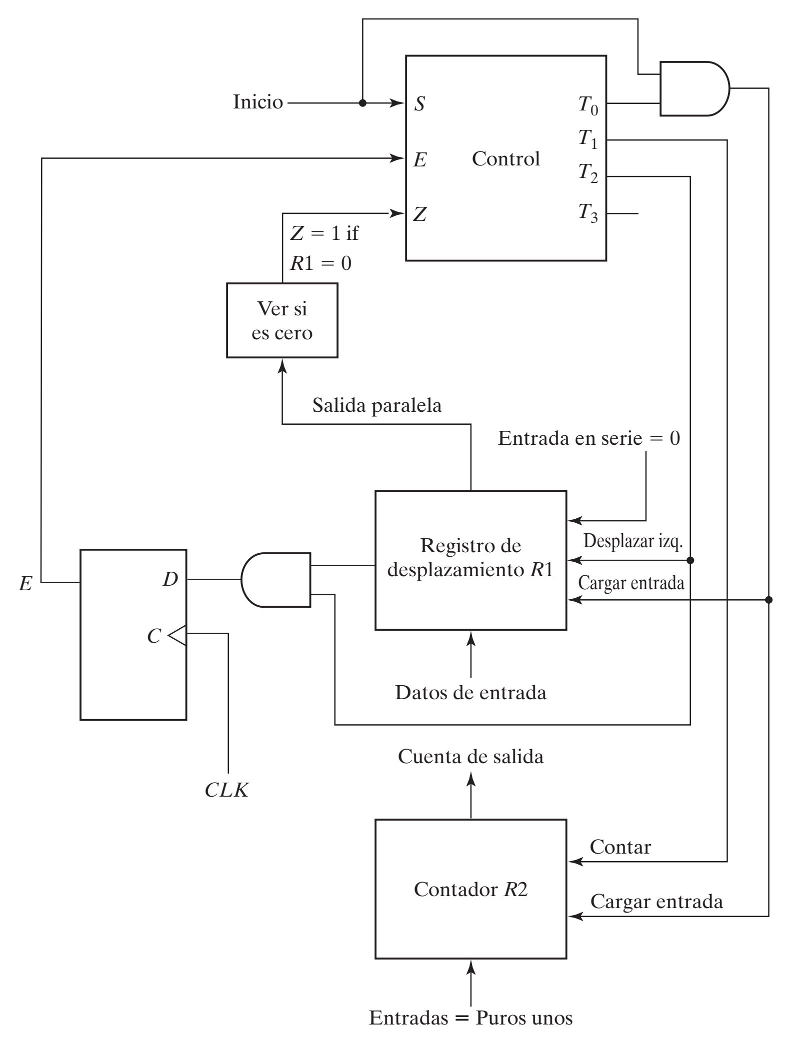


FIGURA 8-22
Diagrama de bloques del contador de unos

El diagrama de bloques del circuito se reproduce en la figura 8-22. El control tiene tres entradas y cuatro salidas. La trayectoria de datos sólo usa tres salidas. El registro *R*1 es de desplazamiento. El registro *R*2 es un contador con carga paralela. Para no complicar el diagrama, no se muestra el reloj, pero se debe aplicar a los dos registros, al flip-flop *E* y a los flip-flops del control.

Las condiciones de entrada del multiplexor para el control se determinan a partir de la tabla 8-8. Las condiciones de entrada se obtienen del diagrama ASM para cada posible transición de estado binario. Se asignan a los cuatro estados los valores binarios 00 a 11. La transición desde el estado actual 00 depende de *S*. La transición desde el estado actual 01 depende de *Z*, y la transición desde el estado actual 11 depende de *E*. El estado 10 pasa al 11 incondicionalmente. Los valores bajo MUX1 y MUX2 en la tabla se determinan a partir de las condiciones booleanas de entrada para el siguiente estado de G_1 y G_0, respectivamente.

Tabla 8-8
Condiciones de entrada de multiplexores para el ejemplo de diseño

Estado actual		Siguiente estado		Condiciones de entrada	Entradas de multiplexores	
G_1	G_0	G_1	G_0		MUX1	MUX2
0	0	0	0	S'		
0	0	0	1	S	0	S
0	1	1	0	Z		
0	1	1	1	Z'	Z'	0
1	0	1	1	Ninguna	1	1
1	1	1	0	E'		
1	1	0	1	E	E'	E

La implementación del control para el ejemplo de diseño se ilustra en la figura 8-23. Se trata de una implementación de tres niveles con los multiplexores en el primer nivel. Las entradas de los multiplexores se obtienen de la tabla 8-8.

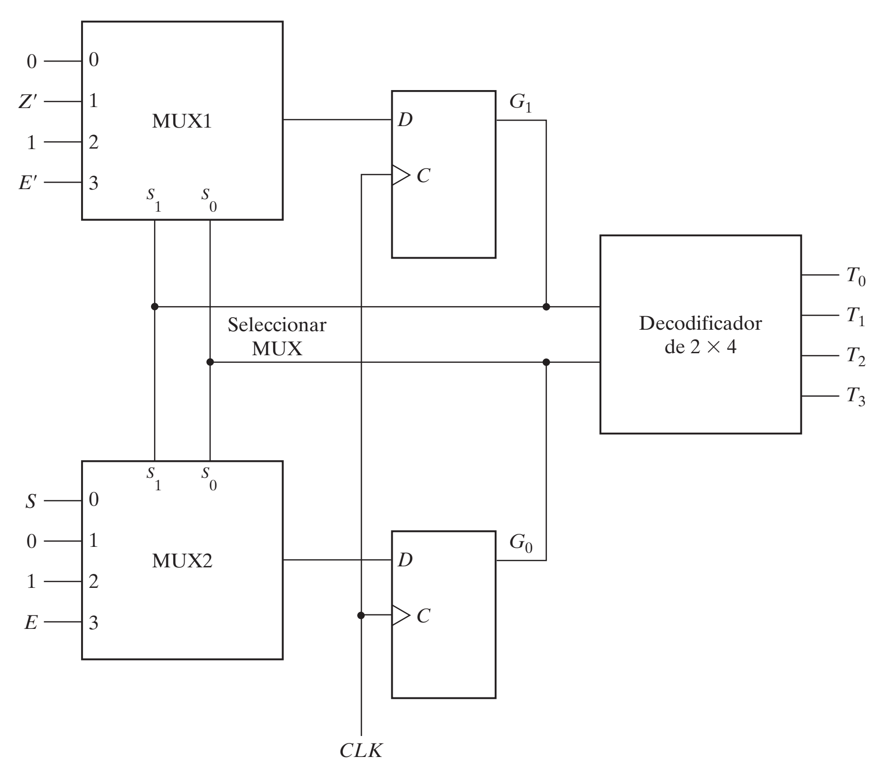


FIGURA 8-23
Implementación del control para el circuito contador de unos

PROBLEMAS

8-1 Explique con palabras las operaciones especificadas por la siguiente notación de transferencia de registros:

a) $R2 \leftarrow R2 + 1, R1 \leftarrow R2$

b) $R3 \leftarrow R3 - 1$

c) If $(T_1 = 1)$ then $(R0 \leftarrow R1)$ else if $(T_2 = 1)$ then $(R0 \leftarrow R2)$

8-2 Dibuje la porción de un diagrama ASM partiendo de un estado inicial. Hay dos señales de control: x y y. Si $xy = 10$, el registro R se incrementa en 1 y el control pasa a un segundo estado. Si $xy = 01$, el registro R se pone en 0 y el control pasa del estado inicial a un tercer estado. En los demás casos, el control permanece en el estado inicial.

8-3 Dibuje los diagramas ASM para las siguientes transiciones de estado:

a) Si $x = 0$, el control pasa del estado T_1 al estado T_2; si $x = 1$, se genera una operación condicional y se pasa de T_1 a T_2.

b) Si $x = 1$, el control pasa de T_1 a T_2 y luego a T_3; si $x = 0$, el control pasa de T_1 a T_3.

c) Se parte del estado T_1; luego, si $xy = 00$, pasar a T_2; si $xy = 01$, pasar a T_3; si $xy = 10$, pasar a T_1; si $xy = 11$, pasar a T_3.

8-4 Muestre en un bloque ASM los ocho trayectorias de salida que emanan de los cuadros de decisión que verifican los ocho posibles valores binarios de tres variables de control: x, y y z.

8-5 Explique en qué difiere un diagrama ASM de un diagrama de flujo convencional. Utilice la figura 8-15 como ejemplo para ilustrar la diferencia en interpretación.

8-6 Construya un diagrama ASM para un sistema digital que cuenta el número de personas en una habitación. Las personas entran por una puerta provista de una fotocelda que cambia una señal x de 1 a 0 cuando la luz se interrumpe, y salen por una segunda puerta provista de una fotocelda similar que cambia una señal y de 1 a 0 cuando la luz se interrumpe. El circuito consta de un contador arriba-abajo con un *display* que indica cuántas personas hay en la habitación.

8-7 Dibuje un diagrama ASM para un circuito con dos registros de ocho bits RA y RB que reciben dos números binarios sin signo y efectúan la operación de resta:

$$RA \leftarrow RA - RB$$

Utilice el método de resta descrito en la sección 1-5 y ponga un flip-flop de préstamo en 1 si la respuesta es negativa.

8-8 Diseñe un circuito digital con tres registros de 16 bits, AR, BR y CR que realicen las operaciones siguientes:

a) Transferir dos números de 16 bits con signo (en representación de complemento a dos) a AR y BR.

b) Si el número que está en AR es negativo, dividirlo entre 2 y transferir el resultado al registro CR.

c) Si el número que está en AR es positivo pero distinto de cero, multiplicar el número que está en BR por 2 y transferir el resultado al registro CR.

d) Si el número que está en AR es cero, poner en cero el registro CR.

8-9 Diseñe el control cuyo diagrama de estados se dio en la figura 8-11a), utilizando un flip-flop por estado (asignación de un solo uno).

8-10 La figura P8-10 corresponde al diagrama de estados de una unidad de control. Tiene cuatro estados y dos entradas, x y y. Dibuje el diagrama ASM equivalente, dejando en blanco los cuadros de estado.

8-11 Diseñe el control cuyo diagrama de estados se presenta en la figura P8-10 utilizando flip-flops *D*.

8-12 Suponga que, en la figura 8-22, *R*1 es el registro de desplazamiento de cuatro bits que se muestra en la figura 6-7. Indique cómo deben conectarse las entradas de desplazamiento y de carga a las entradas s_1 y s_0 del registro de desplazamiento.

8-13 Diseñe el contador de cuatro bits con despeje sincrónico especificado en la figura 8-10.

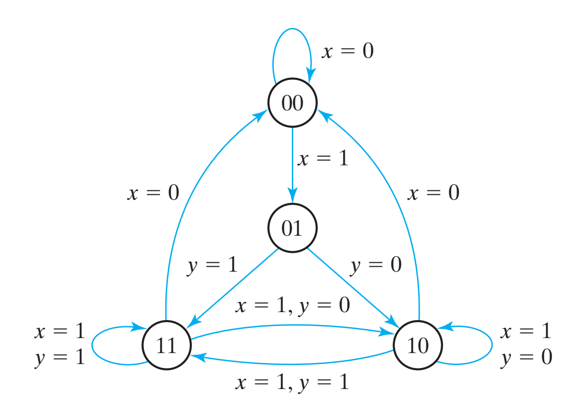


FIGURA P8-10
Diagrama de estados de control para los problemas 8-10 y 8-11

8-14 Diseñe un circuito digital que multiplique dos números binarios por el método de suma repetida. Por ejemplo, para multiplicar 5 × 4, el sistema digital evalúa el producto sumando cuatro veces el multiplicando: 5 + 5 + 5 + 5 = 20. El multiplicando estará en el registro *BR*, el multiplicador estará en *AR* y el producto, en *PR*. Un circuito sumador suma el contenido de *BR* a *PR*. Un circuito para detectar ceros, *Z*, indica cuándo *AR* se vuelve 0 después de decrementarse.

8-15 Demuestre que la multiplicación de dos números de *n* bits da un producto con longitud menor o igual a 2*n* bits.

8-16 En la figura 8-13, el registro *Q* contiene el multiplicador y el registro *B* contiene el multiplicando. Suponga que ambos números son de 16 bits.

a) ¿Cuántos bits cabe esperar en el producto, y dónde quedará?

b) ¿Cuántos bits hay en el contador *P*, y qué número binario se carga en él inicialmente?

c) Diseñe el circuito que verifica si el contador *P* es 0.

8-17 Enumere el contenido de los registros *C*, *A*, *Q* y *P* de forma análoga a como se hizo en la tabla 8-4, durante el proceso de multiplicar los dos números 11111 (multiplicando) y 10101 (multiplicador).

8-18 Calcule el tiempo que toma procesar la operación de multiplicación en el multiplicador binario descrito en la sección 8-6. Suponga que el registro *Q* tiene *n* bits y que el ciclo de reloj es de *t* nanosegundos.

8-19 Diseñe el circuito de control del multiplicador binario especificado por el diagrama de estados de la figura 8-15 utilizando multiplexores, un decodificador y un registro.

8-20 Examine el diagrama ASM de la figura P8-20. No se especifican las operaciones de registros porque sólo nos interesa diseñar la lógica de control.

a) Dibuje el diagrama de estados equivalente.

b) Diseñe el control con un flip-flop por estado.

c) Prepare la tabla de estados para el control.

d) Diseñe el control con tres flip-flops *D*, un decodificador y compuertas.

e) Deduzca una tabla que muestre las condiciones de entrada de multiplexores para el control.

f) Diseñe el control con tres multiplexores, un registro con tres flip-flops y un decodificador de 3 × 8.

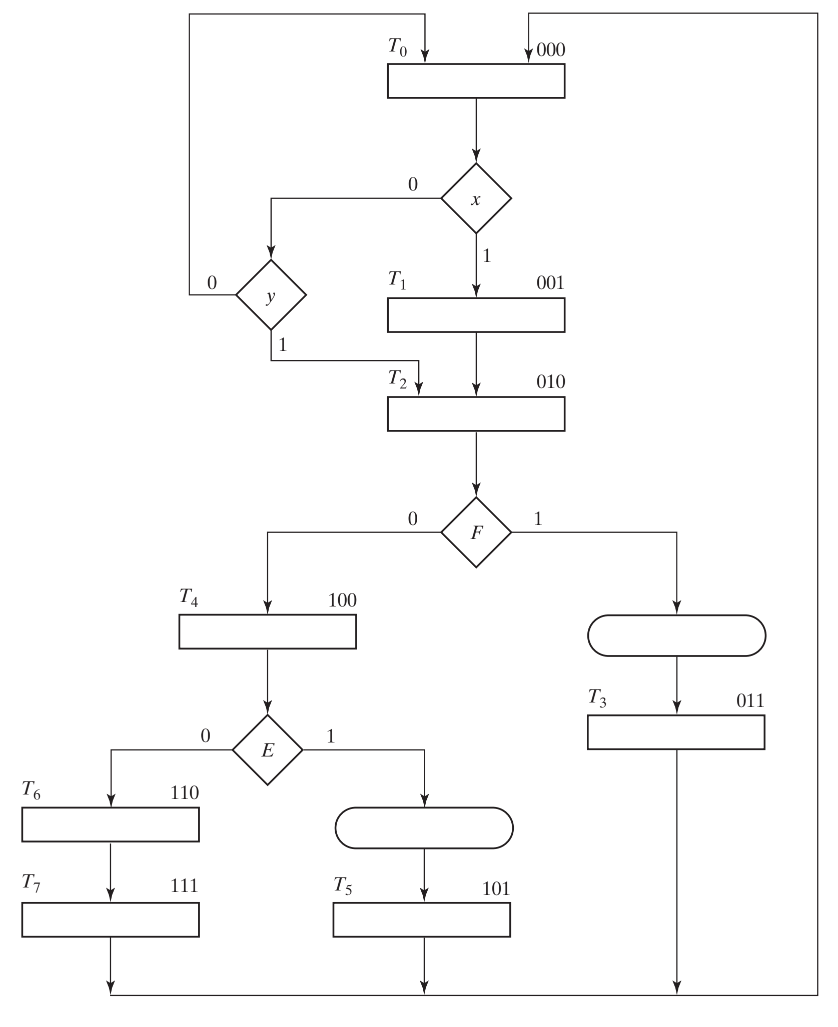


FIGURA P8-20
Diagrama ASM para el problema 8-20

8-21 ¿Qué valor tiene E en cada bloque HDL, suponiendo que $RA = 1$?

a)
```
RA = RA - 1;
if (RA == 0) E = 1;
else E = 0;
```

b)
```
RA <= RA - 1;
if (RA == 0) E <= 1;
else E <= 0;
```

8-22 Utilizando los operadores de Verilog HDL presentados en la tabla 8-1, evalúe el resultado de las operaciones siguientes. Suponga que $A = 4'b0110$, $B = 4'b0010$ y $C = 4'b0000$.

```
A * B; A + B; A - B; ~C; A & B; A | B; A ^ B; & A; ~|C; A || B;
A && C; |A; A < B; A > B; A != B;
```

8-23 Considere este bloque always:

```
always @ (posedge CLK)
if (T1) R1 <= R1 + R2;
else if (T2) R1 <= R1 + 1;
else R1 <= R1;
```

Utilizando un contador de cuatro bits con carga paralela para $R1$ (como en la figura 6-14) y un sumador de cuatro bits, dibuje un diagrama de bloques que muestre las conexiones de los componentes y las señales de control para una posible síntesis del bloque.

8-24 Los sintetizadores lógicos a menudo traducen el enunciado **case** multinivel a multiplexores en hardware. ¿Cómo traduciría el siguiente bloque **case** a hardware? Suponga registros de ocho bits cada uno.

```
case (state)
T0:  R4 = R0;
T1:  R4 = R1;
T2:  R4 = R2;
T3:  R4 = R3;
endcase
```

8-25 En la sección 8-9 se diseñó un circuito que cuenta el número de unos que hay en un registro. El diagrama ASM para ese circuito se muestra en la figura 8-21, y el diagrama de bloques, en la figura 8-22.

a) Escriba la descripción HDL del circuito en el nivel de transferencia de registros.

b) Diseñe la lógica de control empleando un flip-flop por estado (asignación de un solo uno). Dé las ecuaciones de entrada para los cuatro flip-flops.

c) Escriba la descripción estructural HDL del circuito empleando el control diseñado en la parte b) y el diagrama de bloques de la figura 8-22.

d) Escriba un conjunto de pruebas para probar el circuito. Simule el circuito para verificar el funcionamiento descrito en los programas tanto RTL como estructural.

8-26 Escriba la descripción estructural HDL del multiplicador diseñado en la sección 8-6. Utilice el diagrama de bloques de la figura 8-13 y el circuito de control de la figura 8-17. Simule el diseño y verifique el funcionamiento con el conjunto de pruebas del ejemplo HDL 8-6.

8-27 La suma de dos números binarios con signo en la representación de magnitud con signo sigue las reglas de la aritmética ordinaria. Si los dos números tienen el mismo signo (ambos positivos o ambos negativos), las dos magnitudes se suman y la suma tiene el signo de los operandos. Si los dos números tienen distinto signo, el de menor magnitud se resta al de mayor magnitud y el resulta-

do tiene el signo del minuendo. Escriba una descripción HDL de comportamiento para la suma de dos números de ocho bits con signo en representación de magnitud con signo. El bit de extrema izquierda de cada número contiene el signo y los otros siete bits contienen la magnitud.

8-28 Escriba la descripción HDL del circuito diseñado en el problema 8-14.

REFERENCIAS

1. Palnitkar, S. 1996. *Verilog HDL: A Guide to Digital Design and Synthesis*. SunSoft Press (Un título Prentice-Hall).
2. Bhasker, J. 1997. *A Verilog HDL Primer*. Allentown, PA: Star Galaxy Press.
3. Bhasker, J. 1998. *Verilog HDL Synthesis*. Allentown, PA: Star Galaxy Press.
4. Thomas, D. E. y P. R. Moorby. 1998. *The Verilog Hardware Description Language.* 4a. ed. Boston: Kluwer Academic Publishers.
5. Ciletti, M. D. 1999. *Modeling, Synthesis and Rapid Prototyping with Verilog HDL*. Upper Saddle River, NJ: Prentice-Hall.
6. Arnold, M. G. 1999. *Verilog Digital Computer Design*. Upper Saddle River, NJ: Prentice-Hall.
7. Smith, D. J. 1996. *HDL Chip Design*. Madison, AL: Doone Publications.
8. 1995. *IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language* (Norma IEEB 1364-1995). Nueva York: The Institute of Electrical and Electronics Engineers.
9. Mano, M. M. 1993. *Computer System Architecture*, 3a. ed. Upper Saddle River, NJ: Prentice-Hall.
10. Mano, M. M. y C. R. Kime. 2000. *Logic and Computer Design Fundamentals*. 2a. ed. Upper Saddle River, NJ: Prentice-Hall.
11. Hayes, J. P. 1993. *Introduction to Digital Logic Design*. Reading, MA: Addison-Wesley.
12. Clare, C. R. 1971. *Designing Logic Systems Using State Machines*. Nueva York: McGraw-Hill.
13. Winkler, D. y F. Prosser. 1987. *The Art of Digital Design*, 2a. ed. Englewood Cliffs, NJ: Prentice-Hall.

9 Lógica secuencial asincrónica

9-1 INTRODUCCIÓN

Un circuito secuencial se especifica con una sucesión temporal de entradas, salidas y estados internos. En los circuitos secuenciales sincrónicos, el cambio de estado interno se da en respuesta a los pulsos de reloj sincronizados. Los circuitos secuenciales asincrónicos no utilizan pulsos de reloj. El cambio de estado interno se da cuando hay un cambio en las variables de entrada. Los elementos de memoria de los circuitos secuenciales sincrónicos son flip-flops que operan con reloj. Los elementos de memoria en los circuitos secuenciales asincrónicos son flip-flops sin reloj, o bien, elementos de retardo. La capacidad de memoria de un dispositivo de retardo se debe al tiempo finito que la señal tarda en propagarse a través de las compuertas digitales. Muchos circuitos secuenciales asincrónicos semejan circuitos combinacionales con retroalimentación.

Es más difícil diseñar circuitos secuenciales asincrónicos que sincrónicos debido a los problemas de temporización causados por el camino de retroalimentación. En un sistema sincrónico debidamente diseñado, los problemas de temporización se eliminan al disparar todos los flip-flops con el borde del pulso. El cambio de un estado al siguiente se da durante el breve lapso de la transición del pulso. Puesto que los circuitos asincrónicos no usan reloj, el estado del sistema puede cambiar inmediatamente después de que cambian las entradas. Es preciso cuidar que cada nuevo estado mantenga al circuito en una condición estable, aunque exista un camino de retroalimentación.

Los circuitos secuenciales asincrónicos tienen diversas aplicaciones. Se les utiliza cuando la velocidad de operación es importante, sobre todo en casos en que el sistema digital debe responder rápidamente, sin esperar un pulso de reloj. Su uso es más económico en sistemas pequeños independientes que sólo requieren unos cuantos componentes. En tales casos, no resulta práctico incurrir en el gasto que implica incluir un circuito para generar pulsos de reloj. Los circuitos asincrónicos también son útiles en aplicaciones en las que las señales de entrada del sistema podrían cambiar en cualquier momento, con independencia de un reloj interno. La comunicación entre dos unidades, cada una de las cuales tiene su propio reloj autónomo, debe efectuarse con circuitos asincrónicos. Los diseñadores digitales a menudo crean sistemas mixtos

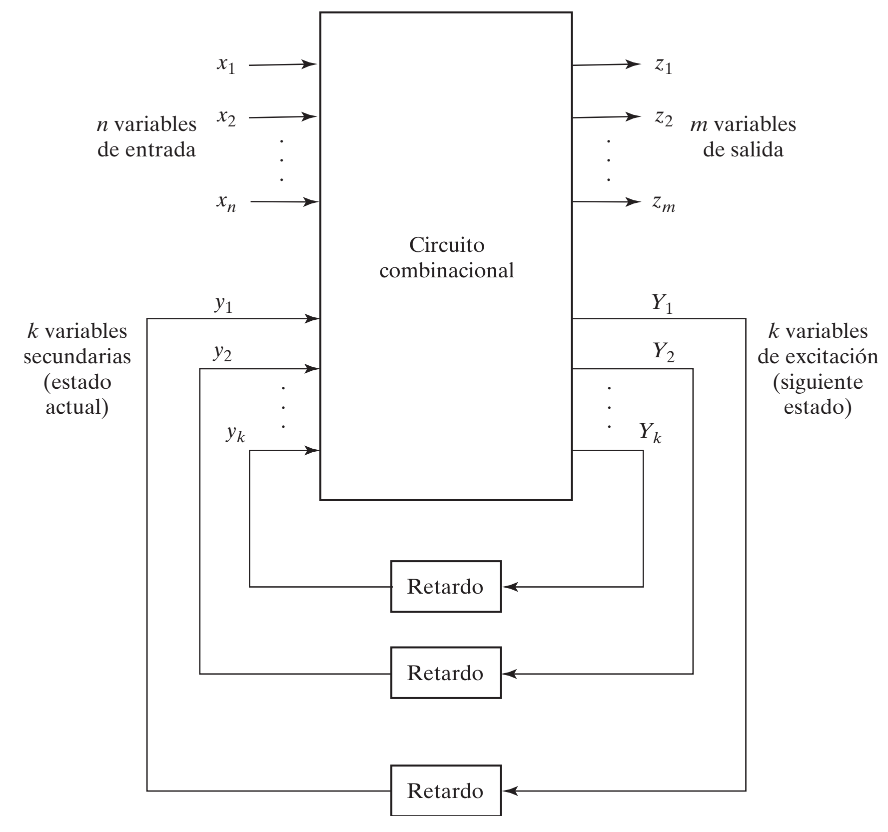


FIGURA 9-1
Diagrama de bloques de un circuito secuencial asincrónico

en los que una parte del sistema sincrónico posee las características de los circuitos asincrónicos. Conocer el comportamiento de la lógica secuencial asincrónica nos ayuda a verificar que el sistema digital total esté funcionando correctamente.

La figura 9-1 presenta el diagrama de bloques de un circuito secuencial asincrónico. Consiste en un circuito combinacional y elementos de retardo conectados para formar lazos de retroalimentación. Hay n variables de entrada, m variables de salida y k estados internos. Los elementos de retardo pueden verse como una memoria a corto plazo para el circuito secuencial. En un circuito de compuertas, el retardo de propagación que hay en la trayectoria de circuito combinacional desde la entrada hasta la salida proporciona un retardo suficiente en el lazo de retroalimentación, y es innecesario insertar elementos específicos de retardo en el lazo. Las variables de estado actual y siguiente estado de los circuitos secuenciales asincrónicos suelen denominarse variables secundarias y variables de excitación, respectivamente. No debemos confundir las variables de excitación con la tabla de excitación que se usa en el diseño de los circuitos secuenciales con reloj.

Cuando una variable de entrada cambia de valor, las y variables secundarias no cambian instantáneamente. Se requiere cierto tiempo para que la señal se propague desde las terminales de entrada, a través del circuito combinacional, hasta las variables de excitación Y, donde

se generan nuevos valores para el siguiente estado. Estos valores se propagan a través de los elementos de entrada y se convierten en el nuevo estado actual de las variables secundarias. Cabe señalar la distinción entre las y y las Y. En la condición de estado estable, son iguales, no así durante la transición. Para un valor dado de las variables de entrada, el sistema está estable si el circuito alcanza una condición de estado estacionario con $y_i = Y_i$ para $i = 1, 2, \ldots, k$. De lo contrario, el circuito estará en una transición continua y decimos que es inestable. Es importante darse cuenta de que sólo se da una transición de un estado estable a otro en respuesta a un cambio en una variable de entrada. Esto contrasta con los sistemas sincrónicos, en los que las transiciones de estado se dan en respuesta a la aplicación de un pulso de reloj.

Para garantizar un funcionamiento correcto, debe permitirse que los circuitos secuenciales asincrónicos alcancen un estado estable antes de cambiar sus entradas a un nuevo valor. Debido al retardo en los conductores y en los circuitos de compuerta, es imposible hacer que dos o más variables de entrada cambien exactamente al mismo tiempo sin que haya incertidumbre respecto a cuál cambió primero. Por ello, generalmente se prohiben los cambios simultáneos de dos o más variables. Esta restricción implica que sólo una variable de entrada puede cambiar de valor en un momento dado, y que el tiempo entre dos cambios de entrada debe ser más largo que el tiempo que el circuito tarda en alcanzar un estado estable. Este tipo de funcionamiento se define como *modo fundamental*. El funcionamiento en modo fundamental supone que las señales de entrada cambian una por una, y sólo cuando el circuito está en condición estable.

9-2 PROCEDIMIENTO DE ANÁLISIS

El análisis de circuitos secuenciales asincrónicos consiste en obtener una tabla o diagrama que describa la sucesión de estados internos y salidas en función de los cambios en las variables de entrada. Un diagrama lógico manifiesta un comportamiento de circuito secuencial asincrónico si tiene uno o más lazos de retroalimentación o si incluye flip-flops sin reloj. En esta sección se analizará el comportamiento de los circuitos secuenciales asincrónicos que tienen trayectorias de retroalimentación sin usar flip-flops. Los flip-flops sin reloj se llaman latches, y en la siguiente sección se explicará su uso en los circuitos secuenciales asincrónicos.

Se presentará el procedimiento de análisis con la ayuda de tres ejemplos específicos. El primero introduce la tabla de transición. El segundo ejemplo define la tabla de flujo. El tercero investiga la estabilidad de los circuitos secuenciales asincrónicos.

Tabla de transición

En la figura 9-2 se presenta un ejemplo de circuito secuencial asincrónico que sólo tiene compuertas. El diagrama muestra claramente dos lazos de retroalimentación desde las salidas de las compuertas OR hacia las entradas de las compuertas AND. El circuito tiene una variable de entrada x y dos estados internos. Los estados internos tienen dos variables de excitación, Y_1 y Y_2, y dos variables secundarias, y_1 y y_2. El retardo asociado a cada lazo de retroalimentación proviene del retardo de propagación entre cada entrada y y su salida correspondiente Y. Cada compuerta lógica en la trayectoria introduce un retardo de propagación de aproximadamente 2 a 10 ns. Los conductores que llevan señales eléctricas introducen un retardo de aproximadamente un nanosegundo por cada 30 cm de cable. Por tanto, no se requieren elementos de retardo externos adicionales si el circuito combinacional y los cables de la trayectoria de retroalimentación proporcionan suficiente retardo.

El análisis del circuito se inicia considerando a las variables de excitación como salidas y a las variables secundarias como entradas. Luego se deducen las expresiones booleanas para

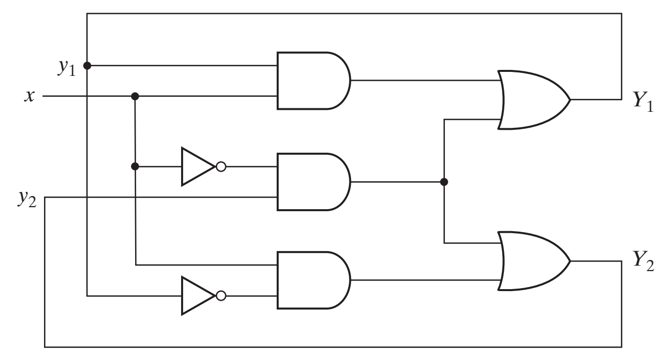
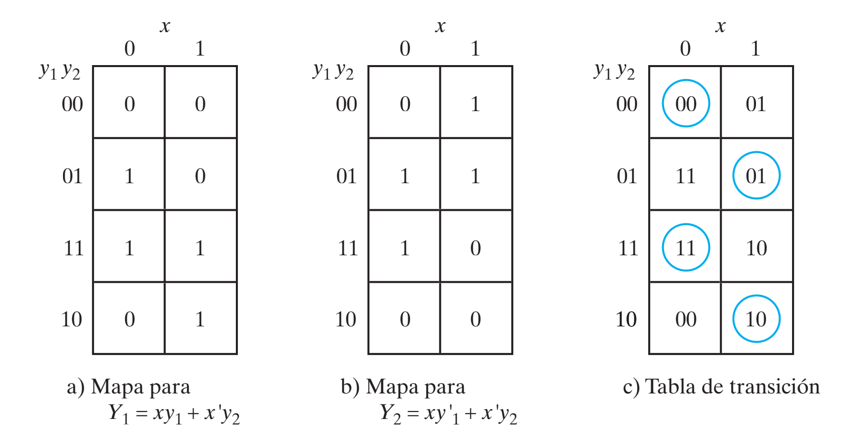


FIGURA 9-2
Ejemplo de circuito secuencial asincrónico

las variables de excitación en función de las variables de entrada y las variables secundarias. Dichas expresiones se obtienen fácilmente del diagrama lógico.

$$Y_1 = xy_1 + x'y_2$$
$$Y_2 = xy_1' + x'y_2$$

El siguiente paso es graficar las funciones Y_1 y Y_2 en un mapa, como se muestra en la figura 9-3a) y b). Usamos los valores binarios codificados de las variables *y* para rotular las filas, y la variable de entrada *x*, para designar las columnas. Esta configuración produce un mapa de tres variables un poco diferente del que se usó en capítulos anteriores. No obstante, es un mapa válido, y este tipo de configuración es más conveniente para manejar circuitos secuenciales asincrónicos. Observe que las variables que corresponden a los cuadros apropiados no se anotan a los costados del mapa como en capítulos anteriores.



y_1y_2 \ x	0	1
00	0	0
01	1	0
11	1	1
10	0	1

a) Mapa para $Y_1 = xy_1 + x'y_2$

y_1y_2 \ x	0	1
00	0	1
01	1	1
11	1	0
10	0	0

b) Mapa para $Y_2 = xy'_1 + x'y_2$

y_1y_2 \ x	0	1
00	(00)	01
01	11	(01)
11	(11)	10
10	00	(10)

c) Tabla de transición

FIGURA 9-3
Mapas y tabla de transición para el circuito de la figura 9-2

La tabla de transición de la figura 9-3c) se obtiene de los mapas combinando los valores binarios de cuadrados correspondientes. La tabla de transición muestra el valor de $Y = Y_1Y_2$ dentro de cada cuadrado. El primer bit de Y se obtiene del valor de Y_1, y el segundo, del valor de Y_2 en el mismo cuadrado. Para que un estado sea estable, el valor de Y debe ser igual al de $y = y_1y_2$. Los elementos de la tabla de transición en los que $Y = y$ se han encerrado en círculos para indicar una condición estable. Los elementos no encerrados en círculos representan estados inestables.

Consideremos ahora el efecto de un cambio en la variable de entrada. El cuadrado para $x = 0$ y $y = 00$ en la tabla de transición muestra que $Y = 00$. Puesto que Y representa el siguiente valor de y, es una condición estable. Si x cambia de 0 a 1 cuando $y = 00$, el circuito cambiará el valor de Y a 01. Esto representa una condición inestable temporal porque Y no es igual al valor actual de y. Lo que sucede a continuación es que, tan pronto como la señal se propaga para hacer a $Y = 01$, la trayectoria de retroalimentación del circuito hace que y cambie a 01. Esto se manifiesta en la tabla de transición con una transición de la primera fila ($y = 00$) a la segunda, donde $y = 01$. Ahora que $y = Y$, el circuito llega a una condición estable con una entrada de $x = 1$. En general, si un cambio en la entrada pone al circuito en un estado inestable, el valor de y cambiará (en tanto x no cambie) hasta llegar a un estado estable (encerrado en un círculo). Al aplicar este tipo de análisis al resto de los cuadrados de la tabla de transición, vemos que el circuito repite la sucesión de estados 00, 01, 11, 10 cuando la entrada alterna repetidamente entre 0 y 1.

Advierta la diferencia entre un circuito secuencial sincrónico y uno asincrónico. En un sistema sincrónico, el estado actual está cabalmente especificado por los valores de los flip-flops y no cambia si la entrada cambia cuando el pulso de reloj está inactivo. En un circuito asincrónico, el estado interno puede cambiar inmediatamente después de un cambio en la entrada. Por ello, a veces resulta conveniente combinar el estado interno con el valor de entrada y llamarlo *estado total* del circuito. El circuito cuya tabla de transición se aprecia en la figura 9-3c) tiene cuatro estados totales estables —$y_1y_2x = 000, 011, 110$ y 101— y cuatro estados totales inestables: 001, 010, 111 y 100.

La tabla de transición de un circuito secuencial asincrónico es similar a la tabla de estados que usamos con los circuitos sincrónicos. Si vemos a las variables secundarias como el estado actual, y a las variables de excitación como el siguiente estado, obtendremos la tabla de estados que se muestra en la tabla 9-1. Esta tabla proporciona la misma información que la tabla de transición. Hay una restricción para el caso asincrónico que no aplica al caso sincrónico. En la tabla de transición asincrónica, por lo regular hay al menos un siguiente estado que es igual al valor de estado actual de cada fila. De lo contrario, todos los estados totales de esa fila serán inestables.

Tabla 9-1
Tabla de estados para el circuito de la figura 9-2

Estado actual		Siguiente estado			
		$x = 0$		$x = 1$	
0	0	0	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	1	1	1	1	0

El procedimiento para obtener una tabla de transición a partir del diagrama de circuito de un circuito secuencial asincrónico es el siguiente:

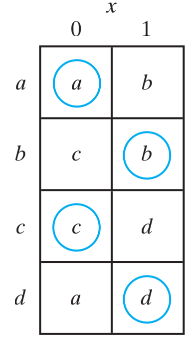
1. Determine todos los lazos de retroalimentación del circuito.
2. Designe la salida de cada lazo de retroalimentación con la variable Y_i, y su entrada correspondiente con y_i, para $i = 1, 2, ..., k$, donde k es el número de lazos de retroalimentación del circuito.
3. Deduzca las funciones booleanas de todas las Y en función de las entradas externas y las y.
4. Grafique cada función Y en un mapa, utilizando las variables y para las filas y las entradas externas para las columnas.
5. Combine todos los mapas en una tabla que tenga el valor de $Y = Y_1 Y_2 \cdots Y_k$ dentro de cada cuadrado.
6. Encierre en círculos los valores de Y que sean iguales al valor de $y = y_1 y_2 \cdots y_k$ de la misma fila.

Una vez que se tenga la tabla de transición, se podrá analizar el comportamiento del circuito observando la transición de estado en función de cambios en las variables de entrada.

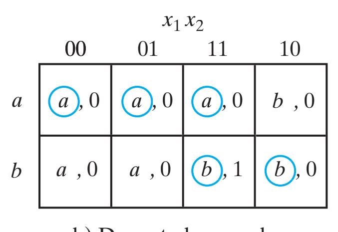
Tabla de flujo

En el diseño de circuitos secuenciales asincrónicos, es más recomendable nombrar los estados con símbolos (letras) sin indicar específicamente sus valores binarios. Una tabla así se denomina *tabla de flujo*, y es similar a una tabla de transición con la salvedad de que los estados internos se simbolizan con letras, no con números binarios. La tabla de flujo también incluye los valores de salida del circuito para cada estado estable.

En la figura 9-4 se muestran ejemplos de tablas de flujo. La de la figura 9-4a) tiene cuatro estados designados con las letras a, b, c y d, y se reduce a la tabla de transición de la figura



a) Cuatro estados con una entrada



b) Dos estados con dos entradas y una salida

FIGURA 9-4
Ejemplos de tablas de flujo

9-3c) si asignamos los valores binarios siguientes a los estados: $a = 00$, $b = 01$, $c = 11$ y $d = 10$. Decimos que la tabla de la figura 9-4a) es una tabla de flujo *primitiva* porque sólo tiene un estado estable en cada fila. La figura 9-4b) constituye una tabla de flujo con más de un estado estable en la misma fila. Tiene dos estados, a y b; dos entradas, x_1 y x_2; y una salida, z. El valor binario de la variable de salida se indica dentro del cuadrado, junto al símbolo de estado y separado de él por una coma. En la tabla de flujo se observa el siguiente comportamiento del circuito. Si $x_1 = 0$, el circuito está en el estado a. Si x_1 cambia a 1 mientras x_2 es 0, el circuito pasa al estado b. Si las entradas son $x_1x_2 = 11$, el circuito podría estar en el estado a o en el b. Si está en a, la salida es 0; si está en b, la salida es 1. El estado b se mantiene si las entradas cambian de 10 a 11. El circuito permanece en el estado a si las entradas cambian de 01 a 11. Recuerde que, en modo fundamental, dos variables de entrada no pueden cambiar simultáneamente, así que no permitimos un cambio de entradas de 00 a 11.

Para obtener el circuito descrito por una tabla de flujo, es necesario asignar a cada estado un valor binario distinto. Esta asignación convierte a la tabla de flujo en una tabla de transición a partir de la cual se deduce el diagrama lógico. Esto se ilustra en la figura 9-5 para la tabla de flujo de la figura 9-4b). Asignamos 0 binario al estado a y 1 binario al estado b. El resultado es la tabla de transición de la figura 9-5a). El mapa de salida de la figura 9-5b) se obtiene directamente de los valores de salida de la tabla de flujo. La función de excitación Y y la función de salida z se simplifican con la ayuda de los dos mapas. El diagrama lógico del circuito aparece en la figura 9-5c).

Este ejemplo ilustra el procedimiento para obtener el diagrama lógico a partir de una tabla de flujo dada. El procedimiento no siempre es tan sencillo como en este ejemplo. La

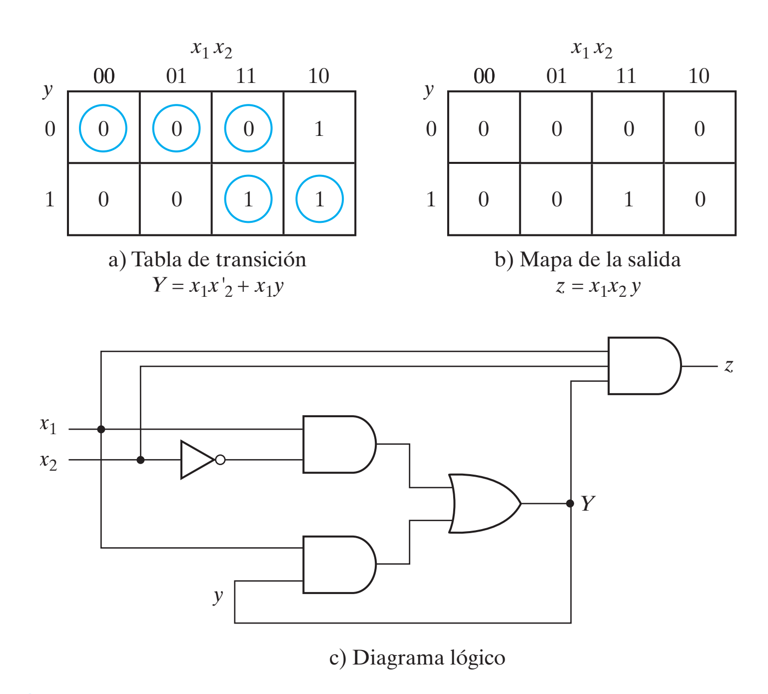


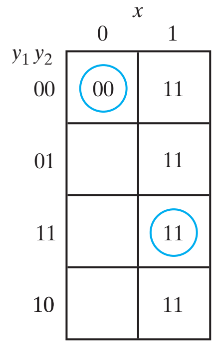
FIGURA 9-5
Deducción del circuito especificado por la tabla de flujo de la figura 9-4b)

asignación de estados binarios y la asignación de salidas a los estados inestables conllevan varias dificultades. Comentaremos los pormenores de esos problemas en las secciones que siguen.

Condiciones de carrera

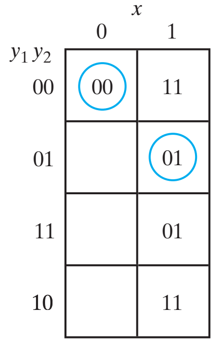
Se dice que existe una condición de *carrera* (*race*) en un circuito secuencial asincrónico cuando dos o más variables de estado binarias cambian de valor en respuesta a un cambio en una variable de entrada. Si los retardos son desiguales, una condición de carrera podría hacer que las variables de estado cambien de manera impredecible. Por ejemplo, si las variables de estado deben cambiar de 00 a 11, la diferencia de retardos podría hacer que la primera variable cambie más rápidamente que la segunda, y el resultado sería que las variables de estado cambiaran sucesivamente de 00 a 10 a 11. Es posible que no se conozca con antelación el orden en que cambian las variables de estado. Si el estado estable final al que llega el circuito no depende del orden en que se modifican las variables de estado, decimos que hay una carrera *no crítica*. Si es posible llegar a dos o más estados estables distintos, dependiendo del orden en que cambian las variables de estado, hay una carrera *crítica*. Para que el funcionamiento sea correcto, es preciso evitar las carreras críticas.

Los dos ejemplos de la figura 9-6 ilustran carreras no críticas. Partimos del estado total estable $y_1y_2x = 000$ y luego cambiamos la entrada de 0 a 1. Las variables de estado deben cambiar de 00 a 11, lo cual define una condición de carrera. Las transiciones que se dan abajo de cada tabla muestran tres posibles formas en que podrían cambiar las variables de estado. Pueden cambiar simultáneamente de 00 a 11, o pueden hacerlo sucesivamente de 00 a 01 a 11, o de 00 a 10 a 11. En todos los casos, el estado estable final es el mismo, o sea que la condición de carrera no es crítica. En a), el estado total final es $y_1y_2x = 111$, y en b), es 011.



a) Posibles transiciones:

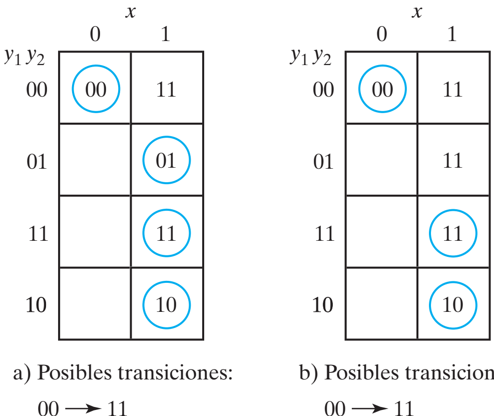
00 → 11
00 → 01 → 11
00 → 10 → 11



b) Posibles transiciones:

00 → 11 → 01
00 → 01
00 → 10 → 11 → 01

FIGURA 9-6
Ejemplos de carreras no críticas



a) Posibles transiciones:

00 → 11
00 → 01
00 → 10

b) Posibles transiciones:

00 → 11
00 → 01 → 11
00 → 10

FIGURA 9-7
Ejemplos de carreras críticas

Las tablas de transición de la figura 9-7 representan carreras críticas. Aquí también partimos del estado total estable $y_1y_2x = 000$ y luego cambiamos la entrada de 0 a 1. Las variables de estado deben cambiar de 00 a 11. Si cambian simultáneamente, el estado total final estable es 111. En la tabla de transición de la parte a), si Y_2 cambia a 1 antes que Y_1 debido a un retardo de propagación desigual, el circuito pasará al estado total estable 011 y permanecerá ahí. En cambio, si Y_1 cambia primero, el estado interno será 10 y el circuito permanecerá en el estado total estable 101. Por tanto, la carrera es crítica porque el circuito pasa a diferentes estados estables dependiendo del orden en que cambian las variables de estado. La tabla de transición de la figura 9-7b) ilustra otra carrera crítica, donde dos posibles transiciones dan como resultado un mismo estado total final, pero la tercera posible transición llega a un estado total distinto.

Es posible evitar carreras efectuando una asignación binaria apropiada a las variables de estado. Es preciso asignar a las variables de estado números binarios de tal manera que sólo una variable de estado pueda cambiar a la vez cuando hay una transición de estado en la tabla de flujo. El tema de la asignación de estados sin carreras se abordará en la sección 9-6.

Podemos evitar carreras haciendo que el circuito pase por estados intermedios inestables con un cambio único de variables de estado. Cuando un circuito pasa por una sucesión única de estados inestables, decimos que tiene un *ciclo*. En la figura 9-8 se ilustran los ciclos. Una vez más, partimos de $y_1y_2 = 00$ y luego cambiamos la entrada de 0 a 1. La tabla de transición de la parte a) da una sucesión *única* que termina en el estado total estable 101. La tabla de b) indica que, aunque las variables de estado cambien de 00 a 11, el ciclo produce una transición única de 00 a 01 y luego a 11. Al usar ciclos, hay que asegurarse de que terminen con un estado estable. Si un ciclo no termina con un estado estable, el circuito seguirá pasando de un estado inestable a otro, y todo el circuito será inestable, como se ilustra en la figura 9-8c) y también en el ejemplo siguiente.

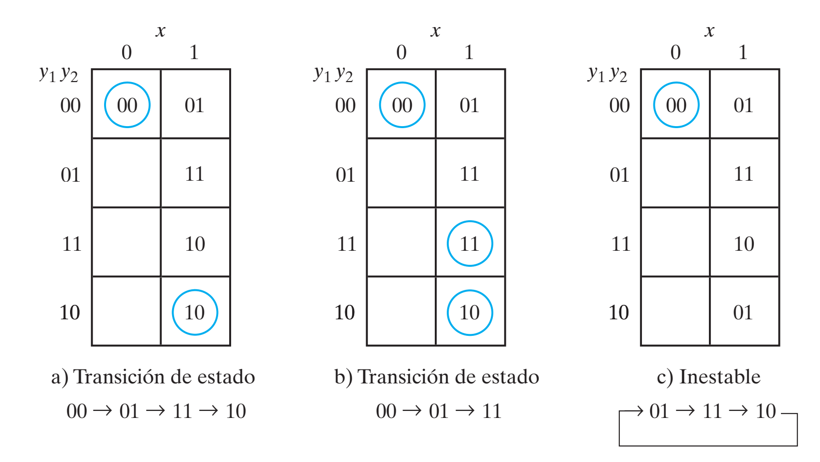


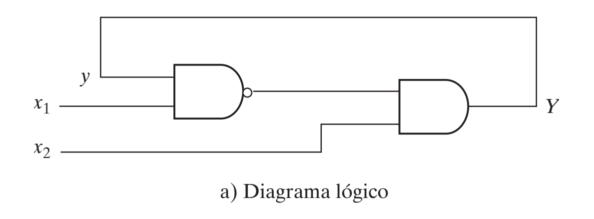
FIGURA 9-8
Ejemplos de ciclos

Consideraciones de estabilidad

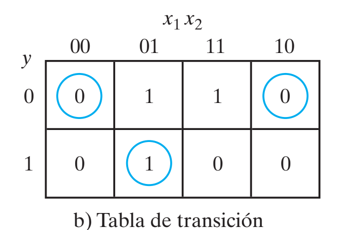
Debido a la conexión de retroalimentación que existe en los circuitos secuenciales asincrónicos, debemos cuidar que el circuito no se vuelva inestable. Una condición inestable hará que el circuito oscile entre estados inestables. El método de análisis de tabla de transición puede ayudarnos a detectar inestabilidades.

Consideremos, por ejemplo, el circuito de la figura 9-9a). La función de excitación es

$$Y = (x_1 y)' x_2 = (x_1' + y') x_2 = x_1' x_2 + x_2 y'$$



a) Diagrama lógico



b) Tabla de transición

FIGURA 9-9
Ejemplo de circuito inestable

La tabla de transición del circuito se reproduce en la figura 9-9b). Los valores de Y que son iguales a y se han encerrado en círculos y representan estados estables. Los valores que no están en círculos indican condiciones inestables. Vemos que la columna 11 no tiene estados estables. Esto implica que, si la entrada x_1x_2 se fija en 11, los valores de Y y y nunca serán iguales. Si $y = 0$, entonces $Y = 1$, y esto causa una transición a la segunda fila de la tabla con $y = 1$ y $Y = 0$. A su vez, esto provoca una transición de vuelta a la primera fila, así que la variable de estado alternará entre 0 y 1 indefinidamente en tanto la entrada sea 11.

La condición de inestabilidad se detecta directamente del diagrama lógico. Sea $x_1 = 1$, $x_2 = 1$ y $y = 1$. La salida de la compuerta NAND es 0, y la de la compuerta AND es 0, lo que hace que Y sea 0, así que $Y \neq y$. Ahora bien, si $y = 0$, la salida de la compuerta NAND será 1, la de la compuerta AND será 1, y Y será 1, de modo que $Y \neq y$. Si suponemos que cada compuerta tiene un retardo de propagación de 5 ns (incluidos los conectores), veremos que Y es 0 durante 10 ns y 1 durante los 10 ns siguientes. Esto producirá una forma de onda cuadrada con un periodo de 20 ns. La frecuencia de oscilación es el recíproco del periodo y es igual a 50 MHz. A menos que estemos diseñando un generador de onda cuadrada, la inestabilidad que podría presentarse en los circuitos secuenciales asincrónicos es indeseable y debe evitarse.

9-3 CIRCUITOS CON LATCHES

Históricamente, los circuitos asincrónicos se conocieron y usaron antes de que se desarrollaran los circuitos sincrónicos. Los primeros circuitos digitales prácticos se construyeron con relevadores, que son más adaptables a las operaciones asincrónicas. Por ello, el método tradicional para configurar circuitos asincrónicos ha utilizado componentes conectados para formar uno o más lazos de retroalimentación. Cuando los circuitos digitales se construyen con componentes electrónicos, resulta conveniente utilizar el latch *SR* (que presentamos en la sección 5-2) como elemento de memoria. El uso de latches *SR* en circuitos secuenciales asincrónicos produce un patrón ordenado en los diagramas lógicos, en el que se distinguen claramente los elementos de memoria. En esta sección, se analizará el funcionamiento del latch *SR* utilizando la técnica presentada en la sección anterior. Luego mostraremos un procedimiento para implementar circuitos secuenciales asincrónicos con latches *SR*.

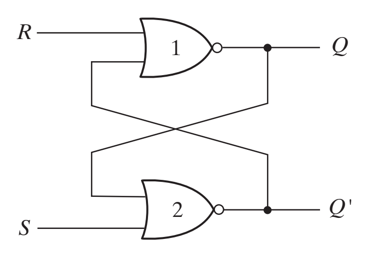
Latch *SR*

El latch *SR* es un circuito digital con dos entradas, S y R, y dos compuertas NOR acopladas en cruz o dos compuertas NAND acopladas en cruz. El circuito de compuertas NOR acopladas en cruz se aprecia en la figura 9-10. Este circuito y su tabla de verdad se tomaron de la figura 5-3. Para analizar el circuito con el método de tabla de transición, lo hemos redibujado en la figura 9-10c) para ver el camino de retroalimentación desde la salida de la compuerta 1 hasta la entrada de la compuerta 2. La salida Q equivale a la variable de excitación Y y a la variable secundaria y. La función booleana para la salida es

$$Y = \left[(S + y)' + R\right]' = (S + y)R' = SR' + R'y$$

Al graficar Y como en la figura 9-10d), se obtiene la tabla de transición para el circuito.

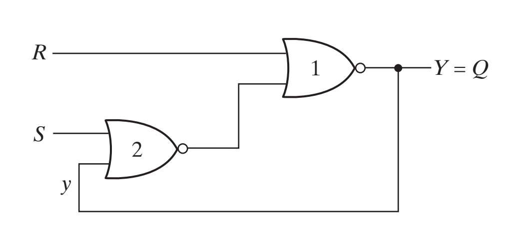
Ahora podemos investigar el comportamiento del latch *SR* utilizando la tabla de transición. Con $SR = 10$, la salida $Q = Y = 1$ y decimos que el latch está encendido o establecido. Si cambiamos S a 0 el circuito permanecerá en el estado encendido. Con $SR = 01$,



a) Circuito acoplado en cruz

S	R	Q	Q'	
1	0	1	0	
0	0	1	0	(Después de $SR = 10$)
0	1	0	1	
0	0	0	1	(Después de $SR = 01$)
1	1	0	0	

b) Tabla de verdad



c) Circuito que muestra retroalimentación

y \ SR	00	01	11	10
0	0	0	0	1
1	1	0	0	1

$Y = SR' + R'y$

$Y = S + R'y$ cuando $SR = 0$

d) Tabla de transición

FIGURA 9-10
Latch *SR* con compuertas NOR

la salida $Q = Y = 0$ y decimos que el latch está apagado o restablecido. Si R vuelve a 0, el circuito permanecerá en el estado apagado. Estas condiciones también se numeran en la tabla de verdad. El circuito presenta ciertos problemas cuando tanto S como R son 1. La tabla de verdad indica que tanto Q como Q' son 0, condición que viola el requisito de que estas dos salidas sean una el complemento de la otra. Además, en la tabla de transición vemos que el paso de $SR = 11$ a $SR = 00$ produce un resultado impredecible. Si S cambia primero a 0, la salida seguirá siendo 0, pero si R cambia primero a 0, la salida pasará a 1. Durante el funcionamiento normal, debemos asegurarnos de que no se aplique 1 a ambas entradas simultáneamente. Esta condición se expresa con la función booleana $SR = 0$, que dice que el AND de S y R siempre debe dar 0.

Volviendo a la función de excitación, vemos que cuando hacemos el OR de la expresión booleana SR' con SR, el resultado es la variable S sola.

$$SR' + SR = S(R' + R) = S$$

De esto, se deduce que $SR' = S$ cuando $SR = 0$. Por tanto, la función de excitación que se dedujo antes,

$$Y = SR' + R'y$$

se expresa así:

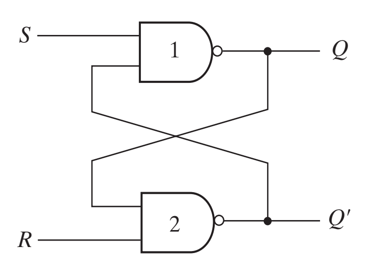
$$Y = S + R'y \qquad \text{cuando } SR = 0$$

Para analizar un circuito con un latch SR, primero hay que verificar que la condición booleana $SR = 0$ se cumpla en todo momento. Entonces usamos la función de excitación reducida para analizar el circuito. No obstante, si se descubre que tanto S como R pueden ser 1 al mismo tiempo, será necesario usar la función de excitación original.

El análisis del latch SR con compuertas NAND se muestra en la figura 9-11. El latch NAND opera con ambas entradas normalmente en 1, a menos que sea preciso cambiar el estado del latch. La aplicación de 0 a R hace que la salida Q cambie a 0, lo cual restablece el latch. Una vez que la entrada R regresa a 1, un cambio de S a 0 produce un cambio al estado encendido. La condición que debe evitarse aquí es que S y R sean 0 simultáneamente. Esta condición se satisface cuando $S'R' = 0$. La función de excitación del circuito es

$$Y = [S(Ry)']' = S' + Ry$$

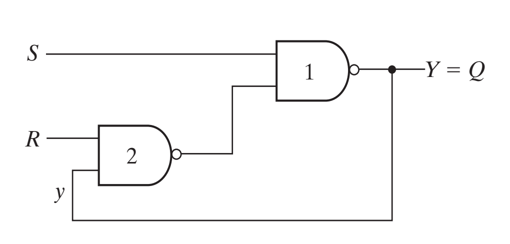
Al compararla con la función de excitación del latch NOR, vemos que S ha sido sustituida por S', y R', por R. Por tanto, las variables de entrada del latch NAND requieren los valores complementados de los que se usan en el latch NOR. Por ello, el latch NAND también se conoce como latch $S'R'$ (o latch $\bar{S}$–$\bar{R}$).



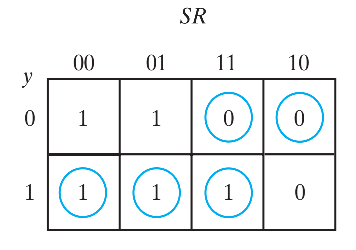
a) Circuito acoplado en cruz

S	R	Q	Q'	
1	0	0	1	
1	1	0	1	(Después de SR = 10)
0	1	1	0	
1	1	1	0	(Después de SR = 01)
0	0	1	1	

b) Tabla de verdad



c) Circuito que muestra retroalimentación



d) Tabla de transición

FIGURA 9-11
Latch *SR* con compuertas NAND

Ejemplo de análisis

Es posible construir circuitos secuenciales asincrónicos utilizando latches *SR* con o sin trayectorias de retroalimentación externas. Desde luego, siempre hay un lazo de retroalimentación dentro del latch mismo. Ilustraremos el análisis de circuitos con latches utilizando un ejemplo específico. A partir de él, podremos generalizar los pasos procedimentales necesarios para analizar otros circuitos similares.

El circuito de la figura 9-12 tiene dos latches *SR* con salidas Y_1 y Y_2. Hay dos entradas, x_1 y x_2, y dos lazos de retroalimentación externos que dan pie a las variables secundarias y_1 y y_2. Observe que este circuito se parece a un circuito secuencial convencional en el que los latches se comportan como flip-flops sin pulsos de reloj. El análisis del circuito requiere obtener primero las funciones booleanas para las entradas *S* y *R* de cada latch.

$$S_1 = x_1 y_2 \qquad S_2 = x_1 x_2$$
$$R_1 = x_1' x_2' \qquad R_2 = x_2' y_1$$

Luego comprobamos si se satisface la condición $SR = 0$ para garantizar el funcionamiento correcto del circuito:

$$S_1 R_1 = x_1 y_2 x_1' x_2' = 0$$
$$S_2 R_2 = x_1 x_2 x_2' y_1 = 0$$

El resultado es 0 porque $x_1 x_1' = x_2 x_2' = 0$.

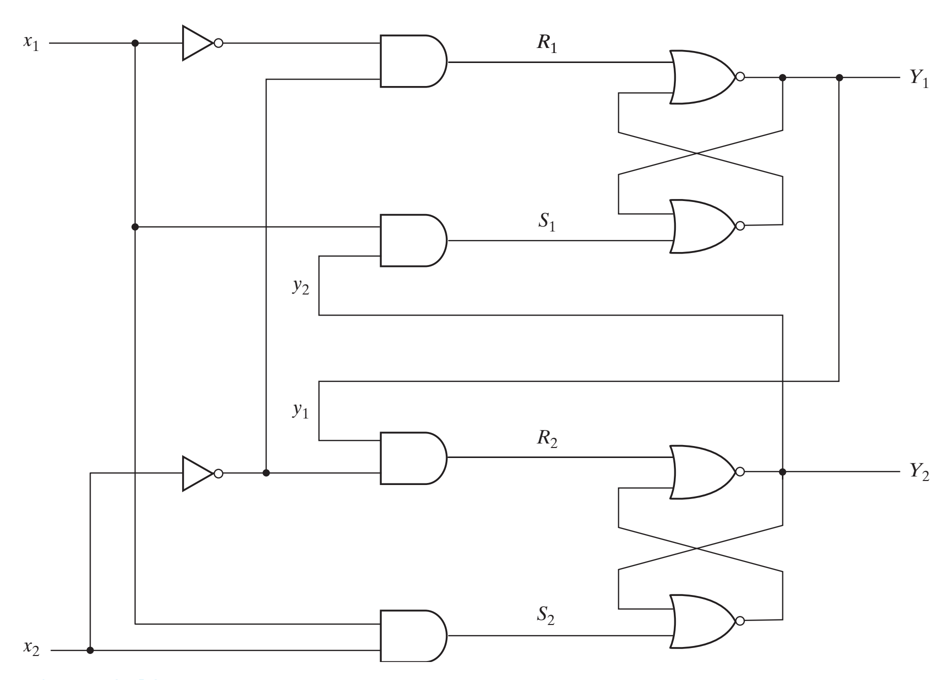


FIGURA 9-12
Ejemplo de circuito con latches *SR*

El siguiente paso es la deducción de la tabla de transición para el circuito. Recuerde que la tabla de transición especifica el valor de Y en función de y y x. Las funciones de excitación se deducen de la relación $Y = S + R'y$.

$$Y_1 = S_1 + R_1'y_1 = x_1y_2 + (x_1 + x_2)y_1 = x_1y_2 + x_1y_1 + x_2y_1$$
$$Y_2 = S_2 + R_2'y_2 = x_1x_2 + (x_2 + y_1')y_2 = x_1x_2 + x_2y_2 + y_1'y_2$$

Ahora preparamos un mapa compuesto para $Y = Y_1Y_2$. Las variables y se asignan a las filas del mapa, y las x, a las columnas, como se indica en la figura 9-13. Usamos las funciones booleanas de Y_1 y Y_2 así expresadas para elaborar el mapa compuesto de Y. Los valores de Y de cada fila que son iguales al valor dado a Y se encierran en un círculo y representan estados estables. Por inspección de la tabla de transición se deduce que el circuito es estable. Hay una condición de carrera crítica cuando el circuito está inicialmente en el estado total $y_1y_2x_1x_2 = 1101$ y x_2 cambia de 1 a 0. Si Y_1 cambia a 0 antes que Y_2, el circuito pasará al estado total 0100 en lugar de 0000. No obstante, si los retardos en las compuertas y latches son aproximadamente iguales, es muy poco probable que se presente esta situación indeseable.

El procedimiento para analizar un circuito secuencial asincrónico con latches SR se resume así:

1. Rotule cada salida de latch con Y_i, y su trayectoria de retroalimentación externa (si lo hay) con y_i, para $i = 1, 2, \ldots, k$.
2. Deduzca las funciones booleanas para las entradas S_i y R_i de cada latch.
3. Verifique que $SR = 0$ para cada latch NOR, o que $S'R' = 0$ para cada latch NAND. Si no se satisface esta condición, cabe la posibilidad de que el circuito no funcione correctamente.
4. Evalúe $Y = S + R'y$ para cada latch NOR, o $Y = S' + Ry$ para cada latch NAND.
5. Construya un mapa en el que las y representen las filas y las entradas x representen las columnas.
6. Grafique el valor de $Y = Y_1Y_2 \cdots Y_k$ en el mapa.
7. Encierre en círculos todos los estados estables ($Y = y$). El mapa así obtenido será la tabla de transición.

y_1y_2 \ x_1x_2	00	01	11	10
00	(00)	(00)	01	(00)
01	(01)	(01)	11	11
11	00	(11)	(11)	10
10	00	(10)	11	(10)

FIGURA 9-13
Tabla de transición para el circuito de la figura 9-12

Tabla de excitación del latch

La tabla de transición del latch *SR* es útil para analizar el latch y definir su funcionamiento. Especifica la variable de excitación *Y* cuando se conocen la variable secundaria *y* y las entradas *S* y *R*. Durante el proceso de implementación, se cuenta con la tabla de transición del circuito y lo que se desea es averiguar los valores de *S* y *R*. Por ello, se necesita una tabla que numere las entradas requeridas *S* y *R* para cada una de las posibles transiciones de *y* a *Y*. Una lista así se denomina *tabla de excitación*.

La tabla de excitación del latch *SR* aparece en la figura 9-14b). Las primeras dos columnas numeran las cuatro posibles transiciones de *y* a *Y*. Las dos columnas que siguen especifican los valores de entrada requeridos que producirán la transición especificada. Por ejemplo, para que haya una transición de $y = 0$ a $Y = 1$, es necesario garantizar que la entrada $S = 1$ y la entrada $R = 0$. Esto se muestra en la segunda fila de la tabla de transición.

Las condiciones de entrada requeridas para cada una de las cuatro transiciones de la tabla de excitación se deducen directamente de la tabla de transición del latch [figura 9-10d)] después de eliminar la condición inestable $SR = 11$. Por ejemplo, para cambiar de $y = 0$ a $Y = 0$, la tabla de transición indica que *SR* puede ser 00 o 01. Esto implica que *S* debe ser 1 y *R* puede ser 0 o 1. Por tanto, la primera fila de la tabla de excitación indica $S = 0$ y $R = X$, donde *X* es una condición de indiferencia que puede ser 0 o 1.

Ejemplo de implementación

La implementación de un circuito secuencial con latches *SR* es un procedimiento para obtener el diagrama lógico a partir de una tabla de transición dada. El procedimiento requiere obtener las funciones booleanas para las entradas *S* y *R* de cada latch. Luego se obtiene el diagrama lógico dibujando los latches *SR* y las compuertas lógicas que implementan las funciones *S* y *R*. Para ilustrar el procedimiento, repetiremos el ejemplo de implementación de la figura 9-5. El circuito de salida es el mismo y no se repetirá.

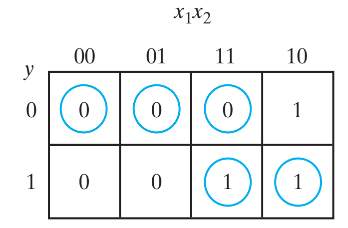
Hemos copiado la tabla de transición de la figura 9-5a) en la figura 9-14a). De la información dada en la tabla de transición, y de las condiciones de la tabla de excitación para el latch presentada en la figura 9-14b), se obtienen los mapas para las entradas *S* y *R* del latch, representados en las figuras 9-14c) y d). Por ejemplo, el cuadrado de la segunda fila, tercera columna ($yx_1x_2 = 111$) en la figura 9-14a) requiere una transición de $y = 1$ a $Y = 1$. La tabla de excitación especifica $S = X$, $R = 0$ para este cambio. Por tanto, el cuadrado correspondiente del mapa *S* se marca con una *X*, y el del mapa *R*, con 0. Todos los demás cuadrados se llenan con valores de forma similar. Luego se usan los mapas para deducir las funciones booleanas simplificadas

$$S = x_1x_2' \qquad \text{y} \qquad R = x_1'$$

El diagrama lógico consiste en un latch *SR* y las compuertas requeridas para implementar las funciones booleanas *S* y *R*. El circuito, utilizando un latch NOR, se reproduce en la figura 9-14e). Si se usa un latch NAND, hay que usar los valores complementados para *S* y *R*.

$$S = (x_1x_2')' \qquad \text{y} \qquad R = x_1$$

Este circuito se ilustra en la figura 9-14f).



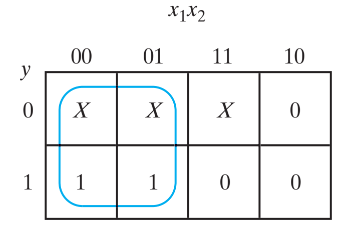
a) Tabla de transición
$Y = x_1x'_2 + x_1y$

y	Y	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	1

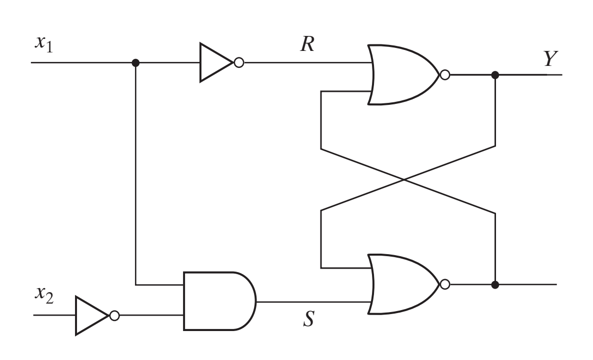
b) Tabla de excitación del latch

y \ x_1x_2	00	01	11	10
0	0	0	0	1
1	0	0	X	X

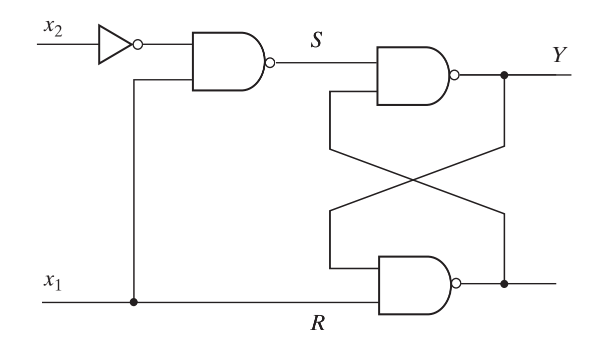
c) Mapa para $S = x_1x'_2$



d) Mapa para $R = x'_1$



e) Circuito con latch NOR



f) Circuito con latch NAND

FIGURA 9-14
Deducción de un circuito de latch a partir de una tabla de transición

Ahora ya podemos resumir el procedimiento general para implementar un circuito con latches SR a partir de una tabla de transición:

1. Dada una tabla de transición que especifica la función de excitación $Y = Y_1Y_2 \cdots Y_k$, deduzca un par de mapas para S_i y R_i, para cada $i = 1, 2, \ldots, k$. Esto se hace utilizando las condiciones especificadas en la tabla de excitación del latch de la figura 9-14b).
2. Deduzca las funciones booleanas simplificadas para S_i y R_i. Se debe tener cuidado de no hacer S_i y R_i iguales a 1 en el mismo cuadrado de minitérmino.

3. Dibuje el diagrama lógico empleando k latches, junto con las compuertas necesarias para generar las funciones booleanas S y R. En el caso de latches NOR, se utilizan las funciones booleanas S y R obtenidas en el paso 2. En el caso de latches NAND, se usan los complementos de los valores obtenidos en el paso 2.

En la sección 9-7 se dará otro ejemplo útil de implementación con latches (figura 9-38).

Circuito antirrebote

La información binaria de entrada de un sistema digital se puede generar manualmente con interruptores mecánicos. Una posición del interruptor suministra un voltaje equivalente al 1 lógico y la otra posición suministra un voltaje distinto equivalente al 0 lógico. También se utilizan interruptores mecánicos para iniciar, parar o restablecer el sistema digital. Al probar circuitos digitales en el laboratorio, las señales de entrada normalmente provienen de interruptores. Una característica que tienen todos los interruptores mecánicos es que, cuando se mueve la palanca de una posición a la otra, el contacto del interruptor vibra o rebota varias veces antes de quedar inmóvil en su posición final. En un interruptor típico, el rebote del contacto podría tardar varios milisegundos en extinguirse. Esto podría hacer que la señal oscilara entre 1 y 0, porque el contacto del interruptor está vibrando.

Un circuito antirrebote elimina la serie de pulsos debida al rebote del contacto y produce una sola transición uniforme de la señal binaria, de 0 a 1 o de 1 a 0. Un circuito de este tipo consiste en un interruptor de un solo polo y dos posiciones conectado a un latch SR, como se observa en la figura 9-15. El contacto central está conectado a tierra, la cual proporciona una señal equivalente al 0 lógico. Cuando uno de los dos contactos, A o B, no está conectado a tierra a través del interruptor, se comporta como una señal de 1 lógico. A veces se conecta un resistor entre cada contacto y un voltaje fijo para tener una señal firme de 1 lógico. Cuando el interruptor se pasa de la posición A a la posición B y de regreso, las salidas del latch producen un solo pulso, como se muestra: negativo para Q y positivo para Q'. El interruptor por lo regular es de botón, cuyo contacto descansa en la posición A. Cuando se oprime el botón, pasa a la posición B, y cuando se suelta vuelve a la posición A.

El circuito antirrebote funciona como sigue. Cuando el interruptor descansa en la posición A, tenemos la condición $S = 0$, $R = 1$ y $Q = 1$, $Q' = 0$ [véase la figura 9-11b)]. Cuando el interruptor se pasa a la posición B, la conexión a tierra hace que R cambie a 0 mientras S cam-

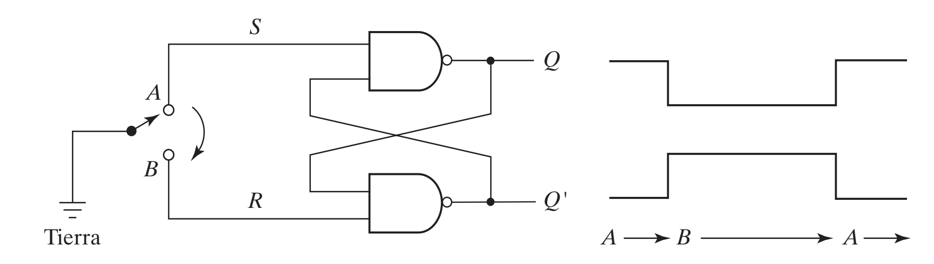


FIGURA 9-15
Circuito antirrebote

bia a 1 porque el contacto A está abierto. Esta condición hace que la salida Q cambie a 0 y que Q' cambie a 1. Una vez que el interruptor hace su contacto inicial con B, rebota varias veces, aunque debemos suponer que, para funcionar correctamente, no rebota tanto como para volver al punto A. La salida del latch no se verá afectada por el rebote del contacto porque Q' permanece en 1 (y Q permanece en 0) ya sea que $R = 0$ (contacto con tierra) o $R = 1$ (sin contacto con tierra). Cuando el interruptor vuelva a la posición A, S cambiará a 0 y Q volverá a 1. Una vez más, la salida exhibirá una transición suave aunque haya un rebote en el contacto de la posición A.

9-4 PROCEDIMIENTO DE DISEÑO

El diseño de un circuito secuencial asincrónico parte del planteamiento del problema y culmina en un diagrama lógico. Se deben llevar a cabo varios pasos de diseño a fin de reducir al mínimo la complejidad del circuito y producir un circuito estable sin carreras críticas. Someramente, los pasos de diseño son los siguientes. Se obtiene una tabla de flujo primitiva a partir de las especificaciones de diseño. La tabla de flujo se reduce a un número mínimo de estados. Se asignan números binarios a los estados para poder obtener la tabla de transición. De la tabla de transición, deducimos el diagrama lógico en forma de circuito combinacional con retroalimentación o de circuito con latches SR.

Ilustraremos el proceso de diseño con un ejemplo específico. Una vez que el lector domine este ejemplo, le será más fácil entender los pasos de diseño que se numeran al final de la sección. Algunos pasos requieren aplicar procedimientos formales, los cuales se estudiarán más a fondo en las secciones siguientes.

Ejemplo de diseño

Es necesario diseñar un circuito de latch con compuerta que tiene dos entradas, G (compuerta) y D (datos), y una salida, Q. La información binaria presente en la entrada D se transfiere a la salida Q cuando G es 1. La salida Q cambiará como cambia la entrada D en tanto $G = 1$. Cuando G cambie a 0, la información que estaba presente en la entrada D en el momento de la transición se mantendrá en la salida Q. El latch con compuerta es un elemento de memoria que acepta el valor de D cuando $G = 1$ y conserva ese valor después de que G cambia a 0. Una vez que $G = 0$, un cambio en D no alterará el valor de la salida Q.

Tabla de flujo primitiva

Ya definimos una tabla de flujo primitiva como una tabla de flujo en la que sólo hay un estado total estable en cada fila. Recuerde que un estado total consiste en el estado interno combinado con la entrada. La deducción de la tabla de flujo primitiva se facilita si primero tratamos de formar una tabla con todos los estados totales posibles del sistema. Ésta se muestra en la tabla 9-2 para el latch con compuerta. Cada fila de la tabla especifica un estado total, que consiste en una designación de letra para el estado interno y una posible combinación de entrada para D y G. También se indica la salida Q para cada estado total. Comenzamos con los dos estados totales que tienen $G = 1$. Por las especificaciones del diseño, sabemos que $Q = 0$ si $DG = 01$ y $Q = 1$ si $DG = 11$ porque D debe ser igual a Q cuando $G = 1$. Asignamos estas

Tabla 9-2
Estados totales del latch con compuerta

	Entradas		Salida	
Estado	***D***	***G***	***Q***	**Comentarios**
a	0	1	0	$D = Q$ porque $G = 1$
b	1	1	1	$D = Q$ porque $G = 1$
c	0	0	0	Después del estado *a* o *d*
d	1	0	0	Después del estado *c*
e	1	0	1	Después del estado *b* o *f*
f	0	0	1	Después del estado *e*

condiciones a los estados *a* y *b*. Cuando *G* cambia a 0, la salida depende del último valor de *D*. Por tanto, si la transición de *DG* es de 01 a 00 a 10, *Q* deberá mantenerse en 0 porque *D* es 0 en el momento de la transición de 1 a 0 en *G*. Si la transición de *DG* es de 11 a 10 a 00, *Q* deberá mantenerse en 1. Esta información da pie a seis estados totales distintos, como se ve en la tabla. Cabe señalar que en el funcionamiento en modo fundamental no se permiten transiciones simultáneas de dos variables de entrada, digamos de 01 a 10 o de 11 a 00.

La tabla de flujo primitiva para el latch con compuerta se presenta en la figura 9-16. Tiene una fila para cada estado y una columna para cada combinación de entrada. Primero, llenaremos un cuadrado de cada fila perteneciente al estado estable de esa fila. Esto se determina con base en la tabla 9-2. Por ejemplo, el estado *a* es estable y la salida es 0 cuando la entrada es 01. Esta información se introduce en la tabla de flujo en la primera fila y la segunda columna. Los otros cinco estados estables se introducen de forma similar en las columnas de entrada correspondientes, junto con su salida.

DG	00	01	11	10
a	*c* , –	(*a*), 0	*b* , –	– , –
b	– , –	*a* , –	(*b*), 1	*e* , –
c	(*c*), 0	*a* , –	– , –	*d* , –
d	*c* , –	– , –	*b* , –	(*d*), 0
e	*f* , –	– , –	*b* , –	(*e*), 1
f	(*f*), 1	*a* , –	– , –	*e* , –

FIGURA 9-16
Tabla de flujo primitiva

Ahora, dado que no se permite un cambio simultáneo de ambas entradas, colocamos guiones en todas las filas que difieran en dos o más variables de las variables de entrada asociadas al estado estable. Por ejemplo, la primera fila de la tabla de flujo muestra un estado estable con entrada 01. Puesto que sólo una entrada puede cambiar en un momento dado, podrá haber un cambio a 00 o a 11, pero no a 10. Por tanto, ponemos dos guiones en la columna 10 de la fila *a*. Al final, esto dará pie a una condición de indiferencia para el siguiente estado y la salida en este cuadrado. Siguiendo este procedimiento, llenamos un segundo cuadrado de cada fila de la tabla de flujo primitiva.

A continuación, es necesario encontrar valores para otros dos cuadrados en cada fila. Los comentarios de la tabla 9-2 nos ayudarán a deducir la información necesaria. Por ejemplo, el estado *c* está asociado a la entrada 00, y se llega a él desde el estado *a* o el *d*, después de un cambio de entrada. Por tanto, indicamos un estado inestable *c* en la columna 00 y las filas *a* y *d* de la tabla de flujo. La salida se marca con un guión para indicar una condición de indiferencia. La interpretación de esto es que si el circuito está en el estado estable *a* y la entrada cambia de 01 a 00, el circuito primero pasa a un siguiente estado inestable, *c*, lo que cambia el valor de estado actual de *a* a *c* y da pie a una transición a la tercera fila y primera columna de la tabla de flujo. Los valores de estado inestable para los demás cuadrados se determinan de forma similar. Todas las salidas asociadas a estados inestables se marcan con un guión para indicar condiciones de indiferencia. Se explicará con mayor detalle la asignación de valores reales a las salidas después de terminar el ejemplo de diseño.

Reducción de la tabla de flujo primitiva

La tabla de flujo primitiva tiene sólo un estado estable en cada fila. El número de filas de la tabla se reduce si colocamos dos o más estados estables en la misma fila de la tabla de flujo. La acción de agrupar estados estables de diferentes filas en una misma fila se denomina *fusión*. La fusión de varios estados estables en la misma fila implica que la variable binaria de estado que en última instancia se asigna a la fila fusionada no cambiará cuando cambie la variable de entrada. Ello se debe a que, en una tabla de flujo primitiva, la variable de estado cambia cada vez que cambia la entrada, mientras que, en una tabla de flujo reducida, una modificación en la entrada no produce una alteración en la variable de estado si el siguiente estado estable está en la misma fila.

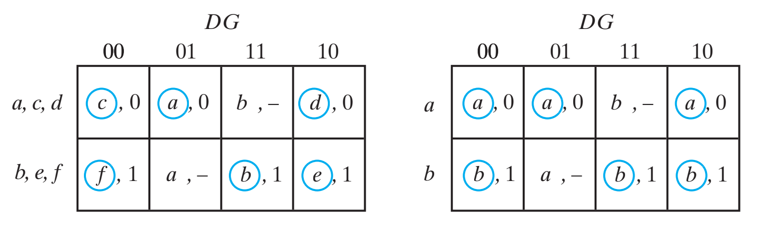
En la siguiente sección se presentará un procedimiento formal para reducir la tabla de flujo. A fin de terminar el ejemplo de diseño sin explicar el procedimiento formal, aplicaremos el proceso de fusión utilizando una versión simplificada de las reglas de fusión. Es posible fusionar dos o más filas de la tabla de flujo primitiva en una fila si no hay conflictos entre los estados y salidas de cada una de las columnas. Siempre que encontremos un símbolo de estado y marcas de indiferencia en la misma columna, anotaremos ese estado en la fila fusionada. Además, si el estado está encerrado en un círculo en una de las filas, también se encerrará en un círculo en la fila fusionada. Se incluye el valor de salida con cada estado estable en la fila fusionada.

Ahora aplicaremos estas reglas a la tabla de flujo primitiva de la figura 9-16. Para mostrar cómo se hace esto, hemos dividido la tabla de flujo en dos partes de tres filas cada una, como se aprecia en la figura 9-17a). En cada parte vemos tres estados estables que pueden fusionarse porque no hay conflictos en ninguna de las cuatro columnas. En la primera columna de la primera parte aparece el estado *c* en todas las columnas y 0 o un guión como salida. Puesto que un guión representa una condición de indiferencia, se le puede asociar a cualquier estado o sa-

	DG 00	01	11	10
a	c, –	(a), 0	b, –	–, –
c	(c), 0	a, –	–, –	d, –
d	c, –	–, –	b, –	(d), 0

	DG 00	01	11	10
b	–, –	a, –	(b), 1	e, –
e	f, –	–, –	b, –	(e), 1
f	(f), 1	a, –	–, –	e, –

a) Estados que son candidatos para fusión



	DG 00	01	11	10
a, c, d	(c), 0	(a), 0	b, –	(d), 0
b, e, f	(f), 1	a, –	(b), 1	(e), 1

	DG 00	01	11	10
a	(a), 0	(a), 0	b, –	(a), 0
b	(b), 1	a, –	(b), 1	(b), 1

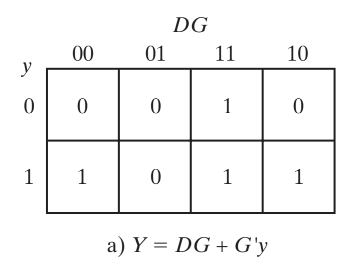
b) Tabla reducida (dos alternativas)

FIGURA 9-17
Reducción de la tabla de flujo primitiva

lida. Los dos guiones de la primera columna se toman como salida 0 para que las tres filas queden idénticas: el estado estable *c* con salida 0. En la segunda columna vemos que es posible asignar valores a los guiones de modo que correspondan a un estado estable *a* con salida 0. Observe que, si el estado está encerrado en un círculo en una de las filas, también se encierra en un círculo en la fila fusionada. De forma similar, la tercera columna puede fusionarse a un estado inestable *b* con salida indiferente, y la cuarta, a un estado estable *d* con salida 0. Así, las tres filas, *a*, *c* y *d*, pueden fusionarse en una sola con tres estados estables y uno inestable, como se indica en la primera fila de la figura 9-17b). La segunda fila de la tabla reducida es resultado de la fusión de las filas *b*, *e* y *f* de la tabla de flujo primitiva. Hay dos formas de dibujar la tabla reducida. Una alternativa es conservar los símbolos de letra de los estados para mostrar la relación entre las tablas de flujo reducida y primitiva. La otra es definir un símbolo de letra común para todos los estados estables de las filas fusionadas. Los estados *c* y *d* se sustituyen por el estado *a*, y los estados *e* y *f* se sustituyen por el estado *b*. En la figura 9-17b) se ilustran ambas alternativas.

Tabla de transición y diagrama lógico

Para obtener el circuito descrito por la tabla de flujo reducida, es preciso asignar a cada estado un valor binario distinto. Esta asignación convierte a la tabla de flujo en una tabla de transición. En el caso general, es necesario efectuar una asignación binaria de estados para garantizar que el circuito no tendrá carreras críticas. El problema de la asignación de estados en circuitos secuenciales asincrónicos, y cómo resolverlo, se tratará en la sección 9-6. Por fortuna, no hay carreras críticas en una tabla de flujo de dos filas, así que podremos terminar el diseño del



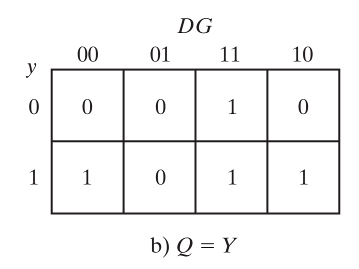


FIGURA 9-18
Tabla de transición y mapa de salida del latch con compuerta

latch con compuerta antes de estudiar la sección 9-6. Si se asigna 0 al estado *a* y 1 al estado *b* de la tabla de flujo reducida de la figura 9-17b), se obtiene la tabla de transición de la figura 9-18a). La tabla de transición es, de hecho, un mapa de la variable de excitación *Y*. A continuación, obtenemos del mapa la función booleana simplificada para *Y*.

$$Y = DG + G'y$$

En la tabla de flujo reducida final hay dos salidas indiferentes. Si asignamos valores a la salida, como se indica en la figura 9-18b), podremos hacer que la salida *Q* sea igual a la función de excitación *Y*. Si asignamos los otros valores posibles a las salidas indiferentes, podremos hacer que la salida *Q* sea igual a *y*. En ambos casos, el diagrama lógico del latch con compuerta es el que se representa en la figura 9-19.

También es posible implementar el diagrama con un latch *SR*. Utilizando el procedimiento delineado en la sección 9-3, primero se obtienen las funciones booleanas para *S* y *R*, como se indica en la figura 9-20a). El diagrama lógico con compuertas NAND se presenta en la figura 9-20b). Observe que el latch con compuerta es un latch *D* sensible al nivel, como el que se presentó en la sección 5-2 y la figura 5-6.

Asignación de salidas a estados inestables

Los estados estables de una tabla de flujo están asociados a valores específicos de salida. Los estados inestables tienen salidas no especificadas indicadas con un guión. Es preciso escoger los valores de salida de los estados inestables de modo que no se presenten salidas falsas mo-

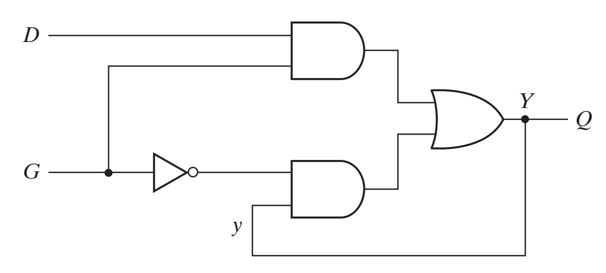
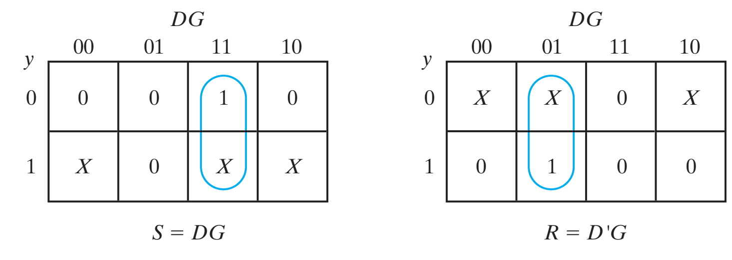
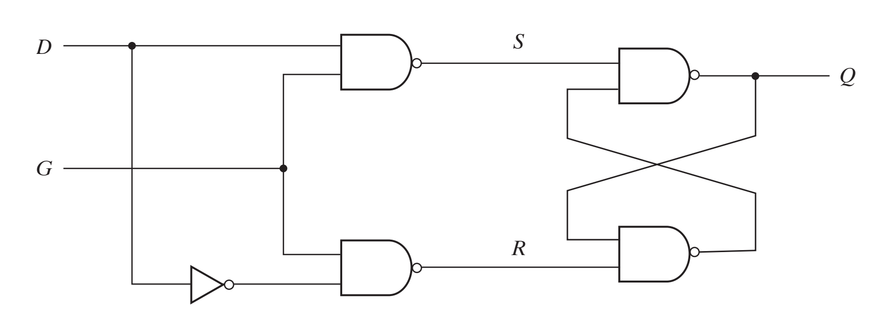


FIGURA 9-19
Diagrama lógico del latch con compuerta



a) Mapas para S y R



b) Diagrama lógico

FIGURA 9-20
Circuito con latch *SR*

mentáneas cuando el circuito conmute entre estados estables. Esto implica que, si no se supone que una variable de salida cambie como resultado de una transición, y hay un estado inestable que es un estado transitorio entre dos estados estables, dicho estado inestable debe tener el mismo valor de salida que los estados estables. Considere, por ejemplo, la tabla de flujo de la figura 9-21a). Una transición del estado estable a al estado estable b pasa por el estado

a	(a), 0	b, –
b	c, –	(b), 0
c	(c), 1	d, –
d	a, –	(d), 1

a) Tabla de flujo

0	0
X	0
1	1
X	1

b) Asignación de salida

FIGURA 9-21
Asignación de valores de salida a estados inestables

inestable *b*. Si la salida asignada al *b* inestable es 1, aparecerá un pulso corto momentáneo en la salida cuando el circuito cambie de una salida 0 en el estado *a* a una salida de 1 en el estado inestable *b* y vuelva a 0 cuando el circuito llegue al estado estable *b*. Por tanto, la salida correspondiente al estado inestable *b* se debe especificar como 0 para evitar una salida falsa momentánea.

Si una variable de salida debe cambiar de valor como resultado de un cambio de estado, se asigna una condición de indiferencia a esa variable. Por ejemplo, la transición del estado estable *b* al estado estable *c* en la figura 9-21a) cambia la salida de 0 a 1. Si se asigna 0 como valor de salida del estado inestable *c*, el cambio en la variable de salida no se dará sino hasta el término de la transición. En cambio, si se asigna 1, el cambio se efectuará al principio de la transición. Puesto que no importa cuándo se efectúe el cambio de salida, colocamos un indicador de indiferencia en la salida asociada al estado inestable *c*. En la figura 9-21b) se presenta la asignación de salida para la tabla de flujo. Esta asignación ilustra las cuatro posibles combinaciones de cambio de salida que pueden presentarse. El procedimiento para efectuar la asignación a salidas asociadas a estados inestables se resume así:

1. Asigne 0 a una variable de salida asociada a un estado inestable que es un estado transitorio entre dos estados estables que tienen 0 en la variable de salida correspondiente.
2. Asigne 1 a una variable de salida asociada a un estado inestable que es un estado transitorio entre dos estados estables que tienen 1 en la variable de salida correspondiente.
3. Asigne una condición de indiferencia a una variable de salida asociada a un estado inestable que es un estado transitorio entre dos estados estables que tienen valores distintos (0 y 1 o 1 y 0) en la variable de salida correspondiente.

Resumen del procedimiento de diseño

El diseño de circuitos secuenciales asincrónicos se efectúa utilizando el procedimiento que se ilustró en el ejemplo anterior. En las secciones siguientes se explicarán algunos de los pasos de diseño que es preciso detallar más. Los pasos del procedimiento son:

1. Obtenga una tabla de flujo primitiva a partir de las especificaciones de diseño dadas. Ésta es la parte más difícil del diseño porque son necesarias intuición y experiencia para interpretar correctamente las especificaciones del diseño.
2. Reduzca la tabla de flujo fusionando filas de la tabla de flujo primitiva. En la sección 9-5 se presenta un procedimiento formal para efectuar esta fusión.
3. Asigne variables binarias de estado a cada fila de la tabla de flujo reducida para obtener la tabla de transición. El procedimiento para asignar estados eliminando cualquier posible carrera crítica se describe en la sección 9-6.
4. Asigne valores de salida a los guiones asociados a los estados inestables, para obtener los mapas de salida. Este procedimiento ya se explicó.
5. Simplifique las funciones booleanas de las variables de excitación y de salida y dibuje el diagrama lógico, como se explicó en la sección 9-2. El diagrama lógico se dibuja utilizando latches *SR*, como se explica en la sección 9-3 y también al final de la sección 9-7.

9-5 REDUCCIÓN DE ESTADOS Y DE TABLAS DE FLUJO

El procedimiento para reducir el número de estados internos de un circuito secuencial asincrónico es similar al procedimiento empleado con circuitos sincrónicos. En la sección 5-6 presentamos un algoritmo para reducir los estados de una tabla de estados totalmente especificada. Repasaremos ese algoritmo y lo aplicaremos a un método de reducción de estados que utiliza una tabla de implicación. Luego modificaremos el algoritmo y la tabla de implicación para cubrir la reducción de estados de tablas de estados incompletamente especificadas. A continuación utilizaremos el algoritmo modificado para explicar el procedimiento de reducción de la tabla de flujo de circuitos secuenciales asincrónicos.

Tabla de implicación

El procedimiento para reducir los estados de tablas de estados totalmente especificadas se basa en el hecho de que dos estados de una tabla de estados se pueden combinar en uno solo si se demuestra que son equivalentes. Dos estados son equivalentes si, para cada posible entrada, dan exactamente la misma salida y pasan a los mismos siguientes estados o a siguientes estados equivalentes. La tabla 6-6 brinda un ejemplo de estados equivalentes que tienen los mismos siguientes estados y las mismas salidas para cada combinación de entradas. Hay ocasiones en que dos estados no presentan los mismos siguientes estados, pero de todas maneras pasan a siguientes estados equivalentes. Considere, por ejemplo, la tabla de estados mostrada en la tabla 9-3. Los estados actuales a y b tienen la misma salida con la misma entrada. Sus siguientes estados son c y d cuando $x = 0$, y b y a cuando $x = 1$. Si se demuestra que el par de estados (c, d) es equivalente, el par de estados (a, b) también será equivalente porque tiene siguientes estados iguales o equivalentes. Cuando existe una relación así, decimos que (a, b) *implican* a (c, d). Asimismo, en las dos últimas filas de la tabla 9-3 vemos que los dos estados (c, d) implican al par de estados (a, b). La característica de los estados equivalentes es que, si (a, b) implican a (c, d), y (c, d) implican a (a, b), entonces ambos pares de estados son equivalentes; es decir, a y b son equivalentes, lo mismo que c y d. Por tanto, las cuatro filas de la tabla 9-3 se reducen a dos filas combinando a y b en un solo estado, y c y d en otro.

Es posible verificar sistemáticamente la equivalencia o no equivalencia de cada par de estados de una tabla con un gran número de estados con la ayuda de una tabla de implicación. La tabla de implicación es un diagrama que consiste en un cuadrado para cada posible par de estados, con espacio para anotar cualesquier estados que pudieran implicarse. Si examinamos

Tabla 9-3
Tabla de estados para ilustrar estados equivalentes

Estado actual	Siguiente estado		Salida	
	x = 0	x = 1	x = 0	x = 1
a	c	b	0	1
b	d	a	0	1
c	a	d	1	0
d	b	d	1	0

Tabla 9-4
Tabla de estados a reducir

Estado actual	Siguiente estado		Salida	
	x = 0	x = 1	x = 0	x = 1
a	*d*	6	0	0
b	*e*	*a*	0	0
c	*g*	*f*	0	1
d	*a*	*d*	1	0
e	*a*	*d*	1	0
f	*c*	*b*	0	0
g	*a*	*e*	1	0

cuidadosamente la tabla, podremos encontrar todos los pares de estados equivalentes. Usaremos la tabla de estados de la tabla 9-4 para ilustrar este procedimiento. La tabla de implicación aparece en la figura 9-22. A la izquierda se enumeran todos los estados definidos en la tabla de estados con excepción del primero, y en la parte de abajo se enumeran todos los estados con excepción del último. Así se obtienen todas las posibles combinaciones de dos estados, y podemos probar la equivalencia de dos estados dados en el cuadrado situado en la intersección de la fila y la columna correspondientes a esos dos estados.

Dos estados no equivalentes se marcan con una cruz (×) en el cuadrado correspondiente, mientras que su equivalencia se indica con una paloma (√). Algunos cuadrados tienen anotados estados implicados que es preciso investigar más a fondo para ver si son equivalentes o no. Los pasos del procedimiento para llenar los cuadrados son los siguientes. Primero, se coloca una cruz en cualquier cuadrado correspondiente a un par de estados cuyas salidas no son igua-

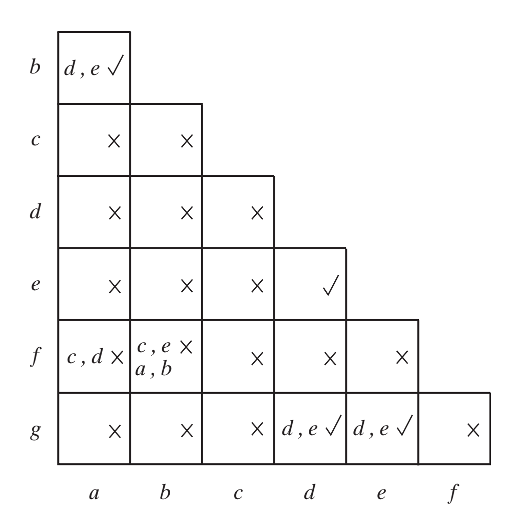


FIGURA 9-22
Tabla de implicación

les con todas las entradas. En este caso, la salida del estado *c* es diferente de la de cualquier otro estado, por lo que se pone una cruz en los dos cuadrados de la fila *c* y en los cuatro cuadrados de la columna *c*. La tabla de implicación tiene otros nueve cuadrados que pertenecen a esta categoría.

A continuación, se anota en los cuadrados restantes los pares de estados implicados por el par de estados que cada cuadrado representa. Comenzamos con el cuadrado superior de la columna izquierda y procedemos hacia abajo; luego continuamos con la siguiente columna hacia la derecha. En la tabla de estados vemos que (a, b) implican a (d, e), así que anotamos a (d, e) en el cuadrado definido por la columna *a* y la fila *b*. Continuamos de esta manera hasta terminar la tabla. Observamos que los estados (d, e) son equivalentes porque pasan al mismo siguiente estado y tienen la misma salida. Por tanto, colocamos una paloma en el cuadrado definido por la columna *d* y la fila *e*, para indicar que los dos estados son equivalentes e independientes de cualquier par implicado.

El siguiente paso consiste en efectuar revisiones sucesivas de la tabla para determinar si hay otros cuadrados que deban marcarse con una cruz. Se tachará un cuadrado de la tabla si contiene al menos un par implicado que no sea equivalente. Por ejemplo, el cuadrado definido por *a* y *f* se marca con una cruz junto a *c*, *d* porque el par (c, d) define un cuadrado que contiene una cruz. Este procedimiento se repite hasta que no es posible tachar más cuadrados. Por último, se marcan con una paloma todos los cuadrados no tachados. Estos cuadrados definen pares de estados equivalentes. En este ejemplo, los estados equivalentes son

$$(a, b) \quad (d, e) \quad (d, g) \quad (e, g)$$

Ahora combinamos pares de estados en grupos mayores de estados equivalentes. Los últimos tres pares se combinan en un juego de tres estados equivalentes (d, e, g) porque cada uno de los estados del grupo es equivalente a los otros dos. La división final de los estados consiste en los estados equivalentes obtenidos de la tabla de implicación y todos los estados restantes de la tabla de estados que no son equivalentes a ningún otro estado.

$$(a, b) \quad (c) \quad (d, e, g) \quad (f)$$

Esto quiere decir que la tabla 9-4 se puede reducir de siete a cuatro estados, uno para cada miembro de la división anterior. Obtenemos la tabla reducida sustituyendo al estado *b* por *a* y a los estados *e* y *g* por *d*. La tabla 9-5 corresponde a la tabla de estados reducida.

Tabla 9-5
Tabla de estados reducida

Estado actual	Siguiente estado		Salida	
	x = 0	**x = 1**	**x = 0**	**x = 1**
a	*d*	*a*	0	0
c	*d*	*f*	0	1
d	*a*	*d*	1	0
f	*c*	*a*	0	0

Fusión de la tabla de estados

Hay ocasiones en que la tabla de estados de un circuito secuencial no está especificada totalmente. Esto sucede cuando ciertas combinaciones de entradas o sucesiones de entradas nunca pueden presentarse debido a restricciones externas o internas. En tales casos, los siguientes estados y salidas que deberían haberse presentado si fueran posibles todas las entradas, nunca se alcanzan y se consideran como condiciones de indiferencia. Aunque a veces es posible representar circuitos secuenciales sincrónicos con tablas de estados incompletamente especificadas, lo que nos interesa aquí son los circuitos secuenciales asincrónicos cuya tabla de flujo primitiva siempre está incompletamente especificada.

Los estados incompletamente especificados pueden combinarse para reducir el número de estados de la tabla de flujo. Tales estados no se consideran equivalentes, porque la definición formal de equivalencia requiere que estén especificadas todas las salidas y todos los siguientes estados para todas las entradas. Más bien, dos estados incompletamente especificados que pueden combinarse se consideran *compatibles*. Dos estados son compatibles si, para cada posible entrada, tienen la misma salida si está especificada, y sus siguientes estados son compatibles si están especificados. Las condiciones de indiferencia marcadas con guiones no tienen efecto alguno en la búsqueda de estados compatibles, pues representan condiciones no especificadas.

El proceso que debe aplicarse para encontrar un grupo adecuado de compatibles con objeto de fusionar una tabla de flujo se divide en tres pasos:

1. Determine todos los pares compatibles empleando una tabla de implicación.
2. Encuentre los compatibles máximos empleando un diagrama de fusión.
3. Encuentre un conjunto mínimo de compatibles que cubra todos los estados y esté cerrado.

Luego se usa el conjunto mínimo de compatibles para fusionar las filas de la tabla de flujo. A continuación, ilustraremos y explicaremos los tres pasos del procedimiento empleando la tabla de flujo primitiva del ejemplo de diseño de la sección anterior.

Pares compatibles

El procedimiento para encontrar pares compatibles se ilustra en la figura 9-23. La tabla de flujo primitiva de a) es igual a la figura 9-16. El contenido de cada cuadro representa el siguiente estado y la salida. Los guiones representan los estados o salidas no especificados. Usamos la tabla de implicación para encontrar estados compatibles de la misma forma que la usamos para hallar estados equivalentes en el caso totalmente especificado. La única diferencia es que, al comparar filas, podemos ajustar los guiones para hacerlos congruentes con cualquier condición deseada.

Dos estados son compatibles si, en todas las columnas de las filas correspondientes de la tabla de flujo, hay estados idénticos o compatibles, y si no hay conflictos entre los valores de salida. Por ejemplo, las filas *a* y *b* de la tabla de flujo son compatibles, pero las filas *a* y *f* sólo serán compatibles si *c* y *f* son compatibles. Sin embargo, las filas *c* y *f* no son compatibles porque tienen salidas distintas en la primera columna. Esta información se registra en la tabla de implicación. Una paloma denota un cuadrado cuyos dos estados son compatibles. Los estados que no son compatibles se marcan con una cruz. En los cuadrados restantes se anotan los pares implicados que es preciso investigar más a fondo.

	00	01	11	10
a	c, –	a, 0	b, –	–, –
b	–, –	a, –	b, 1	e, –
c	c, 0	a, –	–, –	d, –
d	c, –	–, –	b, –	d, 0
e	f, –	–, –	b, –	e, 1
f	f, 1	a, –	–, –	e, –

a) Tabla de flujo primitiva

b	√				
c	√	d, e ×			
d	√	d, e ×	√		
e	c, f ×	√	d, e × c, f ×	×	
f	c, f ×	√	×	d, e × c, f ×	√
	a	b	c	d	e

b) Tabla de implicación

FIGURA 9-23
Tablas de flujo y de implicación

Una vez llenada la tabla de implicación inicial, se examina otra vez para tachar los cuadrados cuyos estados implicados no son compatibles. Los cuadrados restantes con paloma definen los pares compatibles. En el ejemplo de la figura 9-23, los pares compatibles son

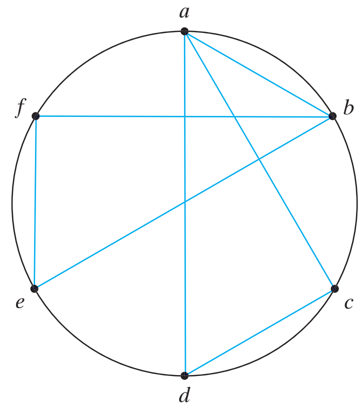
$$(a, b) \quad (a, c) \quad (a, d) \quad (b, e) \quad (b, f) \quad (c, d) \quad (e, f)$$

Máximos compatibles

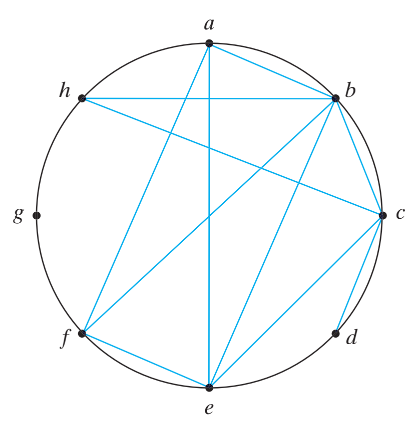
Una vez que se han encontrado todos los pares compatibles, el siguiente paso consiste en hallar conjuntos más grandes de estados compatibles. El *máximo compatible* es un grupo de compatibles que contiene todas las posibles combinaciones de estados compatibles, y puede obtenerse de un diagrama de fusión, como se indica en la figura 9-24. El diagrama de fusión es una gráfica en la que cada estado se representa con un punto colocado sobre la circunferencia de un círculo. Se trazan líneas entre cualesquier dos puntos que formen un par compatible. Podemos deducir todos los posibles compatibles de este diagrama estudiando los patrones geométricos de conexión entre estados. Un punto aislado representa un estado que no es compatible con ningún otro. Una línea representa un par compatible. Un triángulo constituye un compatible de tres estados. Un compatible de n estados se representa en el diagrama de fusión con un polígono de n lados que tiene todas sus diagonales conectadas.

El diagrama de fusión de la figura 9-24a) se obtiene de la lista de pares compatibles deducida de la tabla de implicación de la figura 9.23. Hay siete líneas rectas que conectan puntos, una por cada par compatible. Las líneas forman un patrón geométrico que consiste en dos triángulos que conectan (a, c, d) y (b, e, f) y una línea (a, b). Los máximos compatibles son

$$(a, b) \quad (a, c, d) \quad (b, e, f)$$



a) Máximo compatible:
$(a, b,)$ (a, c, d) (b, e, f)



b) Máximo compatible:
(a, b, e, f) (b, c, h) (c, d) (g)

FIGURA 9-24
Diagramas de fusión

La figura 9-24b) muestra el diagrama de fusión de una tabla de flujo de ocho estados. Los patrones geométricos son un rectángulo con sus dos diagonales conectadas para formar el compatible de cuatro estados (a, b, e, f), un triángulo (b, c, h), una línea (c, d) y un estado solitario g que no es compatible con ningún otro. Los máximos compatibles son

$$(a, b, e, f) \quad (b, c, h) \quad (c, d) \quad (g)$$

Podemos usar el conjunto máximo compatible para fusionar la tabla de flujo asignando una fila de la tabla reducida a cada miembro del conjunto. Sin embargo, es muy común que los máximos compatibles no necesariamente constituyan el conjunto mínimo de compatibles. En muchos casos, es posible encontrar un conjunto más pequeño de compatibles que satisfaga la condición para fusionar filas.

Condición de cobertura cerrada

La condición que debe satisfacerse para fusionar filas es que el conjunto de compatibles escogido debe *cubrir* todos los estados y debe ser *cerrado*. El conjunto cubre todos los estados si incluye todos los estados de la tabla de estados original. La condición de cierre se satisface si no hay estados implicados o si los estados implicados están incluidos en el conjunto. Un conjunto cerrado de compatibles que cubre todos los estados se denomina *cobertura cerrada*. Explicaremos la condición de cobertura cerrada con dos ejemplos.

Consideremos los máximos compatibles de la figura 9-24a). Si eliminamos (a, b), nos quedará un conjunto de dos compatibles:

$$(a, c, d) \quad (b, e, f)$$

Los seis estados de la tabla de flujo de la figura 9-23 están incluidos en este conjunto. Esto satisface la condición de cobertura. No hay estados implicados para (a, c), (a, d), (c, d), (b, e), (b, f) ni (e, f), como se ve en la tabla de implicación de la figura 9-23b), así que también se satisface la condición de cierre. Por tanto, la tabla de flujo primitiva se puede fusionar en dos

filas, una para cada uno de los compatibles. La construcción detallada de la tabla reducida para este ejemplo específico se efectuó en la sección anterior y se muestra en la figura 9-17b).

El segundo ejemplo proviene de una tabla de flujo primitiva (no se muestra) cuya tabla de implicación se presenta en la figura 9-25a). Los pares de compatibles deducidos de la tabla de implicación son

$$(a, b)\ (a, d)\ (b, c)\ (c, d)\ (c, e)\ (d, e)$$

Determinamos los máximos compatibles con el diagrama de fusión de la figura 9-25b):

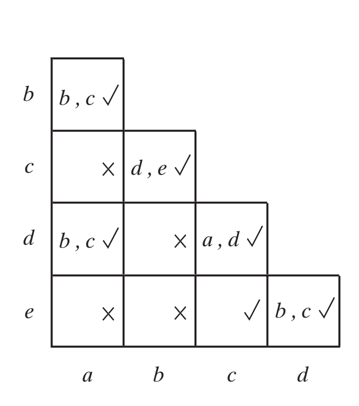
$$(a, b) \quad (a, d) \quad (b, c) \quad (c, d, e)$$

Si escogemos los dos compatibles

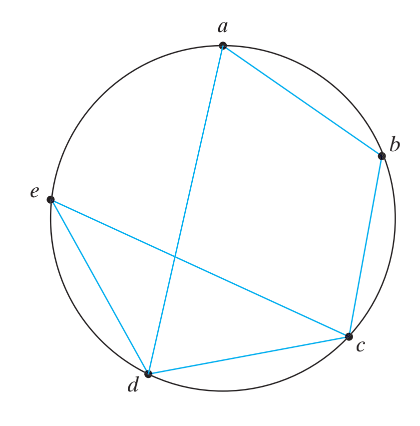
$$(a, b) \quad (c, d, e)$$

el conjunto cubrirá los cinco estados de la tabla original. La condición de cierre se verifica utilizando una tabla de cierre, que aparece en la figura 9-25c). Los pares implicados que se indican para cada compatible se toman directamente de la tabla de implicación. Los estados implicados para (a, b) son (b, c). Sin embargo, (b, c) no está incluido en el conjunto escogido de (a, b) (c, d, e), así que este conjunto de compatibles no es cerrado. Un conjunto de compatibles que satisface la condición de cobertura cerrada es

$$(a, d) \quad (b, c) \quad (c, d, e)$$



a) Tabla de implicación



b) Diagrama de fusión

Compatibles	(a, b)	(a, d)	(b, c)	(c, d, e)
Estados implicados	(b, c)	(b, c)	(d, e)	(a, d,) (b, c,)

c) Tabla de cierre

FIGURA 9-25
Selección de un conjunto de compatibles

El conjunto está cubierto porque contiene los cinco estados. Cabe señalar que los estados se pueden repetir. La condición de cierre se satisface porque los estados implicados son (*b*, *c*), (*d*, *e*) y (*a*, *d*), que están incluidos en el conjunto. La tabla de flujo original (no se muestra aquí) se reduce de cinco a tres filas fusionando las filas *a* y *d*, *b* y *c*, *d* y *e*. Otros compatibles que satisfacen la condición de cobertura cerrada son (*a*, *b*) (*b*, *c*) (*d*, *e*). En general, podría haber más de una manera de fusionar filas al reducir una tabla de flujo primitiva.

9-6 ASIGNACIÓN DE ESTADO SIN CARRERAS

Una vez que se ha deducido una tabla de flujo reducida para un circuito secuencial asincrónico, el siguiente paso del diseño es asignar variables binarias a cada estado estable. Esta asignación transforma la tabla de flujo en su tabla de transición equivalente. El objetivo primario al escoger una asignación apropiada de estados binarios es evitar las carreras críticas. Ya ilustramos el problema de las carreras críticas en la sección 9-2 con la figura 9-7.

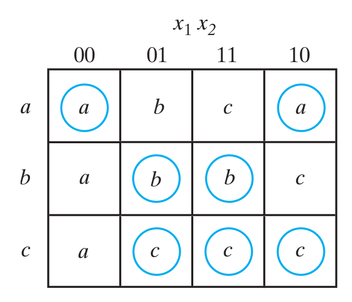
Es posible evitar carreras críticas efectuando una asignación de estados binarios tal que sólo una variable cambie en un momento dado cuando se efectúa una transición de estado en la tabla de flujo. Para lograrlo, es necesario que los estados entre los que hay transiciones reciban asignaciones adyacentes. Decimos que dos valores binarios son adyacentes si sólo difieren en una variable. Por ejemplo, 010 y 011 son adyacentes porque sólo difieren en el tercer bit.

Para garantizar que una tabla de transición no tenga carreras críticas, es necesario probar cada posible transición entre dos estados estables y comprobar que las variables de estado binarias cambien una a la vez. Éste es un proceso tedioso, sobre todo si la tabla tiene muchas filas y columnas. Para simplificarlo, explicaremos el procedimiento de asignación de estados binarios utilizando ejemplos que sólo tienen tres y cuatro filas en la tabla de flujo. Estos ejemplos ilustrarán el procedimiento general que debe seguirse para garantizar una asignación de estados sin carreras. Ese procedimiento podrá aplicarse a tablas de flujo con cualquier cantidad de filas y columnas.

Ejemplo de tabla de flujo de tres filas

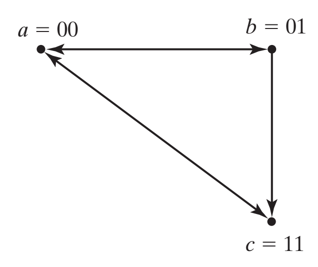
La asignación de una sola variable binaria a una tabla de flujo de dos filas no da pie a problemas de carrera crítica. Una tabla de flujo con tres filas requiere la asignación de dos variables binarias. La asignación de valores binarios a los estados estables podría causar carreras críticas si no se efectúa correctamente. Consideremos, por ejemplo, la tabla de flujo reducida de la figura 9-26a). Se han omitido las salidas por sencillez. Una inspección de la fila *a* revela que hay una transición del estado *a* al estado *b* en la columna 01 y del estado *a* al estado *c* en la columna 11. Esta información se transfiere a un *diagrama de transición*, como se aprecia en la figura 9-26b). Las flechas de *a* a *b* y de *a* a *c* representan las dos transiciones que acabamos de mencionar. Asimismo, las transiciones de las otras dos filas se representan con flechas en el diagrama de transición. Este diagrama es una representación gráfica de todas las transiciones requeridas entre las filas.

Para evitar carreras críticas, debemos encontrar una asignación de estados binarios tal que sólo una variable binaria cambie durante cada transición de estado. En el diagrama de transición se muestra un intento por encontrar semejante asignación. Se asigna 00 binario al estado *a*, y 11 binario al estado *c*. Esta asignación causará una carrera crítica durante la transición de *a* a *c* porque hay dos cambios en las variables de estado binarias. Vemos que la transición de *c* a *a* también causa una condición de carrera, pero ésta no es crítica.



	x_1x_2 00	01	11	10
a	*a*	*b*	*c*	*a*
b	*a*	*b*	*b*	*c*
c	*a*	*c*	*c*	*c*

a) Tabla de flujo



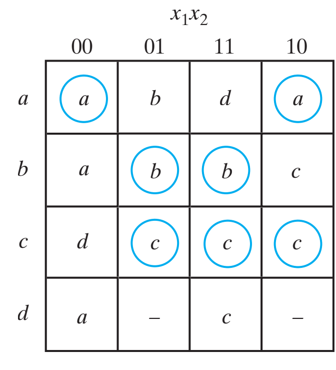
b) Diagrama de transición

FIGURA 9-26
Ejemplo de tabla de flujo de tres filas

Podemos obtener una asignación sin carreras si añadimos otra fila a la tabla de flujo. El uso de una cuarta fila no aumenta el número de variables de estado binarias, pero sí permite la formación de ciclos entre dos estados estables. Consideremos la tabla de flujo modificada de la figura 9-27. Las primeras tres filas representan las mismas condiciones que la tabla original de tres filas. Asignamos a la cuarta fila, rotulada d, el valor binario 10, que es adyacente tanto a a como a c. Ahora la transición de a a c debe pasar por d, así que las variables binarias cambian de $a = 00$ a $d = 10$ a $c = 11$, con lo que se evita una carrera crítica. Esto se logra cambiando la fila a, columna 11, a d, y la fila d, columna 11, a c. De forma análoga, vemos que la transición de c a a pasa por el estado inestable d, aunque la columna 00 constituye una carrera no crítica.

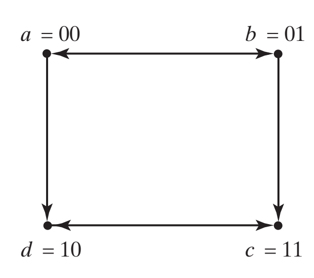
La tabla de transición correspondiente a la tabla de flujo con la asignación indicada de estados binarios se reproduce en la figura 9-28. Los dos guiones de la fila d representan estados no especificados que se consideran condiciones de indiferencia. Sin embargo, hay que tener cuidado de no asignar 10 a estos cuadrados, para evitar la posibilidad de establecer un estado estable indeseable en la cuarta fila.

Este ejemplo ilustra el uso de una fila extra en la tabla de flujo para lograr una asignación sin carreras. La fila extra no se asigna a ningún estado estable específico; más bien, sirve para



	x_1x_2 00	01	11	10
a	*a*	*b*	*d*	*a*
b	*a*	*b*	*b*	*c*
c	*d*	*c*	*c*	*c*
d	*a*	–	*c*	–

a) Tabla de flujo



b) Diagrama de transición

FIGURA 9-27
Tabla de flujo con una fila extra

	x_1x_2 00	01	11	10
$a = 00$	00	01	10	00
$b = 01$	00	01	01	11
$c = 11$	10	11	11	11
$d = 10$	00	–	11	–

FIGURA 9-28
Tabla de transición

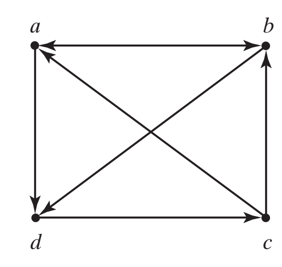
convertir una carrera crítica en un ciclo que pasa por transiciones adyacentes entre dos estados estables. En algunos casos, podría no bastar una fila extra para evitar carreras críticas, y podría ser necesario añadir dos o más filas extras a la tabla de flujo. Ilustraremos esto en el siguiente ejemplo.

Ejemplo de tabla de flujo de cuatro filas

Una tabla de flujo con cuatro filas requiere un mínimo de dos variables de estado. Aunque ocasionalmente es posible una asignación sin carreras con sólo dos variables de estado binarias, en muchos casos la necesidad de filas extra para evitar carreras críticas exigirá el uso de tres variables de estado binarias. Consideremos, por ejemplo, la tabla de flujo y su diagrama de transición correspondiente, que se muestran en la figura 9-29. Si no hubiera transiciones en la dirección diagonal (de *b* a *d* o de *c* a *a*), sería posible encontrar una asignación adyacente para las cuatro transiciones restantes. Con una o dos transiciones diagonales, es imposible asignar dos variables binarias que satisfagan el requisito de adyacencia. Por tanto, se necesitan al menos tres variables de estado binarias.

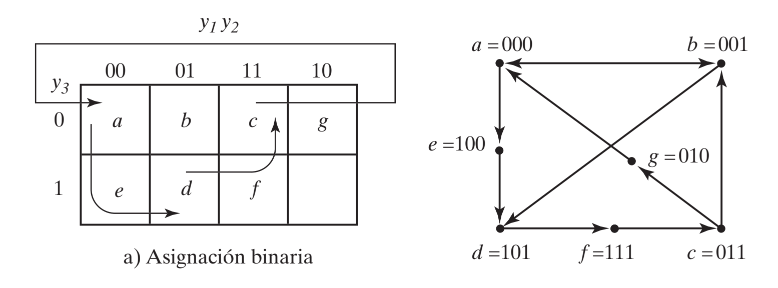
	00	01	11	10
a	*b*	*a*	*d*	*a*
b	*b*	*d*	*b*	*a*
c	*c*	*a*	*b*	*c*
d	*c*	*d*	*d*	*c*

a) Tabla de flujo



b) Diagrama de transición

FIGURA 9-29
Ejemplo de tabla de flujo de cuatro filas



a) Asignación binaria

b) Diagrama de transición

FIGURA 9-30
Selección de filas extra para la tabla de flujo

La figura 9-30 muestra el mapa de asignación de estados apropiado para cualquier tabla de flujo de cuatro filas. Los estados *a*, *b*, *c* y *d* son los originales; *e*, *f* y *g* son estados extra. Los estados colocados en cuadrados adyacentes en el mapa tendrán asignaciones adyacentes. Asignamos 001 al estado *b*, que es adyacente a los otros tres estados originales. La transición de *a* a *d* debe pasar por el estado extra *e* para que produzca un ciclo y sólo una variable binaria cambie a la vez. De forma análoga, se hace que la transición de *c* a *a* pase por *g*, y que la transición de *d* a *c* pase por *f*. Mediante la asignación dada por el mapa, la tabla de cuatro filas se puede expandir a una tabla de siete filas que no tiene carreras críticas, como se observa en la figura 9-31. Aunque la tabla de flujo tiene siete filas, sólo hay cuatro estados estables. Los estados de las tres filas extra, no encerrados en círculos, sólo están ahí para poder efectuar una transición sin carreras entre los estados estables.

	00	01	11	10
000 = *a*	*b*	*a*	*e*	*a*
001 = *b*	*b*	*d*	*b*	*a*
011 = *c*	*c*	*g*	*b*	*c*
010 = *g*	–	*a*	–	–
110 –	–	–	–	–
111 = *f*	*c*	–	–	*c*
101 = *d*	*f*	*d*	*d*	*f*
100 = *e*	–	–	*d*	–

FIGURA 9-31
Asignación de estados a la tabla de flujo modificada

Este ejemplo ilustra una posible forma de seleccionar filas extra en una tabla de flujo para lograr una asignación sin carreras. Un mapa de asignación de estados similar al de la figura 9-30a) es útil en la generalidad de los casos. A veces es posible aprovechar elementos no especificados de la tabla de flujo. En lugar de añadir filas a la tabla, podría ser posible eliminar carreras críticas haciendo que algunas de las transiciones de estado pasen por los estados indiferentes. La asignación real se efectúa por ensayo y error hasta encontrar una asignación satisfactoria que resuelva todas las carreras críticas.

Método de múltiples filas

El método para efectuar asignaciones de estado sin carreras añadiendo filas extra a la tabla de flujo, que ilustramos con los dos ejemplos anteriores, se conoce como método de *filas compartidas*. Existe un segundo método, menos eficiente pero más fácil de aplicar, llamado método de *múltiples filas*. En la asignación de múltiples filas, cada estado de la tabla de flujo original se sustituye

y_1 \ y_2y_3	00	01	11	10
0	a_1	b_1	c_1	d_1
1	c_2	d_2	a_2	b_2

a) Asignación binaria

	00	01	11	10
$000 = a_1$	b_1	(a_1)	d_1	(a_1)
$111 = a_2$	b_2	(a_2)	d_2	(a_2)
$001 = b_1$	(b_1)	d_2	(b_1)	a_1
$110 = b_2$	(b_2)	d_1	(b_2)	a_2
$011 = c_1$	(c_1)	a_2	b_1	(c_1)
$100 = c_2$	(c_2)	a_1	b_2	(c_2)
$010 = d_1$	c_1	(d_1)	(d_1)	c_1
$101 = d_2$	c_2	(d_2)	(d_2)	c_2

b) Tabla de flujo

FIGURA 9-32
Asignación de múltiples filas

por dos o más combinaciones de variables de estado. El mapa de asignación de estados de la figura 9-32a) muestra una asignación de múltiples filas que se puede usar con cualquier tabla de flujo de cuatro filas. Hay dos variables de estado binarias para cada estado estable, siendo una el complemento lógico de la otra. Por ejemplo, el estado original *a* se sustituye por dos estados equivalentes, $a_1 = 000$ y $a_2 = 111$. Los valores de salida, que no se indican aquí, deben ser los mismos en a_1 y a_2. Vemos que a_1 es adyacente a b_1, c_2 y d_1, y a_2 es adyacente a c_1, b_2 y d_2; asimismo, cada estado es adyacente a tres estados designados con letra distinta. El comportamiento del circuito es el mismo si el estado interno es a_1 o a_2, y lo mismo sucede con los demás estados.

La figura 9-32b) exhibe la asignación de múltiples filas para la tabla de flujo original de la figura 9-29a). La tabla expandida se forma sustituyendo cada fila de la tabla original por dos filas. Por ejemplo, la fila *b* se sustituye por las filas b_1 y b_2, y se introduce el estado estable *b* en las columnas 00 y 11 en ambas filas. Una vez que se han introducido todos los estados estables, se introducen los estados inestables consultando la asignación especificada en el mapa de la parte a). Al escoger el siguiente estado para un estado actual dado, se selecciona del mapa un estado adyacente al estado actual. En la tabla original, los siguientes estados de *b* son *a* y *d* para las entradas 10 y 01, respectivamente. En la tabla expandida, los siguientes estados para b_1 son a_1 y d_2 porque éstos son los estados adyacentes a b_1. Asimismo, los siguientes estados para b_2 son a_2 y d_1, porque son adyacentes a b_2.

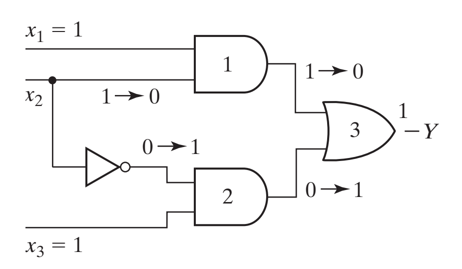
En la asignación de múltiples filas, el cambio de un estado estable a otro siempre causará el cambio de sólo una variable de estado binaria. Cada estado estable tiene dos asignaciones binarias con exactamente la misma salida. En cualquier momento dado, sólo se está usando una de las asignaciones. Por ejemplo, si partimos del estado a_1 y la entrada 01 y luego cambiamos la entrada a 11, 01, 00 y de vuelta a 01, la sucesión de estados internos será a_1, d_1, c_1 y a_2. Aunque el circuito inicia en el estado a_1 y termina en el estado a_2, en lo que a la relación de entrada-salida concierne, los dos estados, a_1 y a_2, son equivalentes al estado *a* de la tabla de flujo original.

9-7 PELIGROS

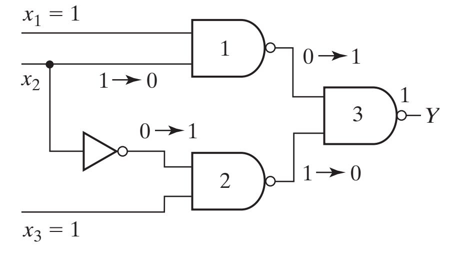
Al diseñar circuitos secuenciales asincrónicos, hay que tener cuidado de respetar ciertas restricciones y precauciones para garantizar un funcionamiento correcto. El circuito se debe operar en modo fundamental, o sea que sólo debe cambiar una entrada en cualquier momento dado, y no debe tener carreras críticas. Además, existe otro fenómeno, llamado *peligro* (*hazard*), que podría hacer que el circuito no funcione correctamente. Los peligros son transitorios de conmutación indeseables que podrían aparecer en la salida de un circuito debido a que diferentes trayectorias tienen diferentes retardos de propagación. Se presentan peligros en los circuitos combinacionales, donde podrían causar un valor temporal de salida falsa. Cuando esta condición se presenta en circuitos secuenciales asincrónicos, podría causar una transición a un estado estable indebido. Por tanto, es necesario cuidar la posibilidad de peligros y determinar si causan operaciones indebidas. Si lo hacen, será preciso tomar medidas para eliminar su efecto.

Peligros en circuitos combinacionales

Un peligro es una condición en la que un cambio en una sola variable produce en la salida un cambio momentáneo que no debería presentarse. El circuito de la figura 9-33a) ilustra un peligro. Supongamos que las tres entradas son 1 inicialmente. Esto hace que la salida de la compuerta 1 sea 1, que la de la compuerta 2 sea 0 y que la del circuito sea 1. Consideremos ahora un cambio de x_2, de 1 a 0. La salida de la compuerta 1 cambia a 0, y la de la compuerta 2, a 1,



a) Circuito AND-OR



b) Circuito NAND

FIGURA 9-33
Circuitos con peligros

con lo que la salida del circuito permanece en 1. Sin embargo, la salida del circuito podría cambiar momentáneamente a 0 si se toma en cuenta el retardo de propagación a través del inversor. El retardo en el inversor podría hacer que la salida de la compuerta 1 cambie a 0 antes de que la salida de la compuerta 2 cambie a 1. En ese caso, las dos entradas de la compuerta 3 serán momentáneamente 0, y la salida del circuito cambiará a 0 durante el breve lapso que la señal de entrada de x_2 se retarda mientras se propaga a través del circuito inversor.

El circuito de la figura 9-33b) es una implementación NAND de la misma función booleana. Tiene un peligro por el mismo motivo. Dado que las compuertas 1 y 2 son NAND, sus salidas son el complemento de las salidas de las compuertas AND correspondientes. Cuando x_2 cambia de 1 a 0, las dos entradas de la compuerta 3 podrían ser 1, lo que haría que la salida produjera un cambio momentáneo a 0 cuando debería haberse mantenido en 1.

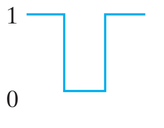
Los dos circuitos que se ilustran en la figura 9-33 implementan la función booleana en suma de productos:

$$Y = x_1x_2 + x_2'x_3$$

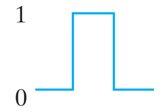
Este tipo de implementación podría hacer que la salida cambie a 0 cuando debería mantenerse en 1. Si el circuito se implementa en producto de sumas (véase la sección 3-5), o sea,

$$Y = (x_1 + x_2')(x_2 + x_3)$$

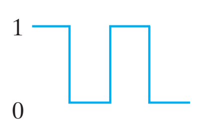
la salida podría cambiar momentáneamente a 1 cuando debería mantenerse en 0. El primer caso se describe como *peligro de 1 estático*, y el segundo, como *peligro de 0 estático*. Un tercer tipo de peligro, llamado *peligro dinámico*, hace que la salida cambie tres o más veces, cuando debería cambiar de 1 a 0 o de 0 a 1. La figura 9-34 ilustra los tres tipos de peligros. Cuando un



a) Peligro de 1 estático

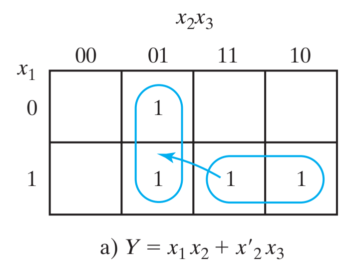


b) Peligro de 0 estático



c) Peligro dinámico

FIGURA 9-34
Tipos de peligros



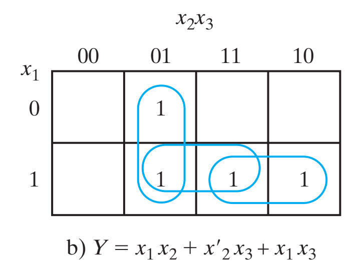


FIGURA 9-35
Mapas que ilustran un peligro y su eliminación

circuito se implementa en suma de productos con compuertas AND-OR o con compuertas NAND, la eliminación del peligro de 1 estático garantiza que no habrá peligros de 0 estático ni peligros dinámicos.

La presencia de un peligro se detecta examinando el mapa del circuito en cuestión. Por ejemplo, consideremos el mapa de la figura 9-35a), que corresponde a la función implementada en la figura 9-33. El cambio en x_2 de 1 a 0 lleva al circuito del minitérmino 111 al minitérmino 101. Hay un peligro porque el cambio en la entrada hace que un término producto diferente cubra los dos minitérminos. El término producto implementado en la compuerta 1 cubre al minitérmino 111 y el término producto implementado en la compuerta 2 cubre al minitérmino 101. Cada vez que el circuito debe pasar de un término producto a otro, cabe la posibilidad de un intervalo momentáneo en el que ninguno de los dos términos sea 1, con lo que surgiría una salida 0 indeseable.

Lo que se hace para eliminar un peligro es encerrar los dos minitérminos en cuestión con otro término producto que traslape ambos grupos. Esto se muestra en el mapa de la figura 9-35b), donde los dos minitérminos que causan el peligro se combinan en un solo término producto. El circuito sin peligros que se obtiene con esta configuración se aprecia en la figura 9-36. La compuerta extra del circuito genera el término producto x_1x_3. En general, los peligros de circuitos combinacionales se eliminan cubriendo cualesquier dos minitérminos que podrían generar un peligro, con un término producto común a ambos. La eliminación de peligros requiere añadir compuertas redundantes al circuito.

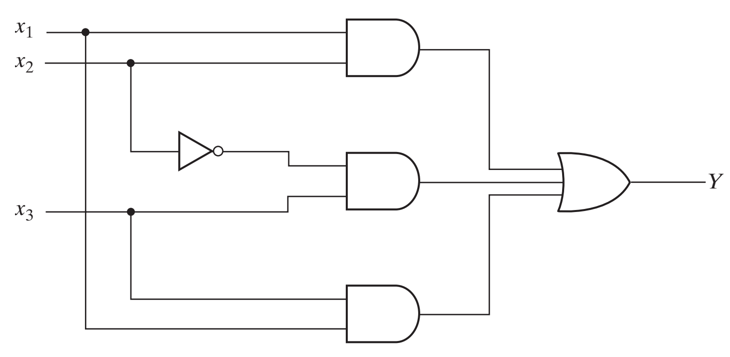


FIGURA 9-36
Circuito sin peligros

Peligros en circuitos secuenciales

En el diseño normal de circuitos combinacionales asociados a circuitos secuenciales sincrónicos, no hay que preocuparse por los peligros, pues las señales erróneas momentáneas generalmente no causan problemas. En cambio, si una señal momentáneamente incorrecta se realimenta a un circuito secuencial asincrónico, podría hacer que el circuito pase a un estado estable indebido. Esto se ilustra en la figura 9-37. Si el circuito está en el estado estable total $yx_1x_2 = 111$ y la entrada x_2 cambia de 1 a 0, el siguiente estado estable total deberá ser 110. Sin embargo, el peligro podría hacer que la salida *Y* cambie a 0 momentáneamente. Si esta señal falsa se realimenta a la compuerta 2 antes de que la salida del inversor cambie a 1, la salida de la compuerta 2 se mantendrá en 0 y el circuito pasará al estado estable total 010, que es incorrecto. Este desperfecto se elimina añadiendo una compuerta extra, como se hizo en la figura 9-36.

Implementación con latches *SR*

Otra forma de evitar los peligros estáticos en circuitos secuenciales asincrónicos es implementar el circuito con latches *SR*. Una señal 0 momentánea aplicada a la entrada *S* o *R* de un latch NOR no afecta el estado del circuito. Asimismo, una señal 1 momentánea aplicada a las entradas *S* y *R* de un latch NAND no afecta el estado del latch. En la figura 9-33b) vimos que una expresión de suma de productos de dos niveles implementada con compuertas NAND podría tener un peligro de 1 estático si ambas entradas de la compuerta 3 cambian a 1, lo que haría que la salida del circuito cambie momentáneamente de 1 a 0. Sin embargo, si la compuerta 3 forma parte de un latch, la señal 1 momentánea no afectará la salida porque una tercera entrada de la compuerta provendrá del lado complementado del latch, el cual será 0, y por ello la

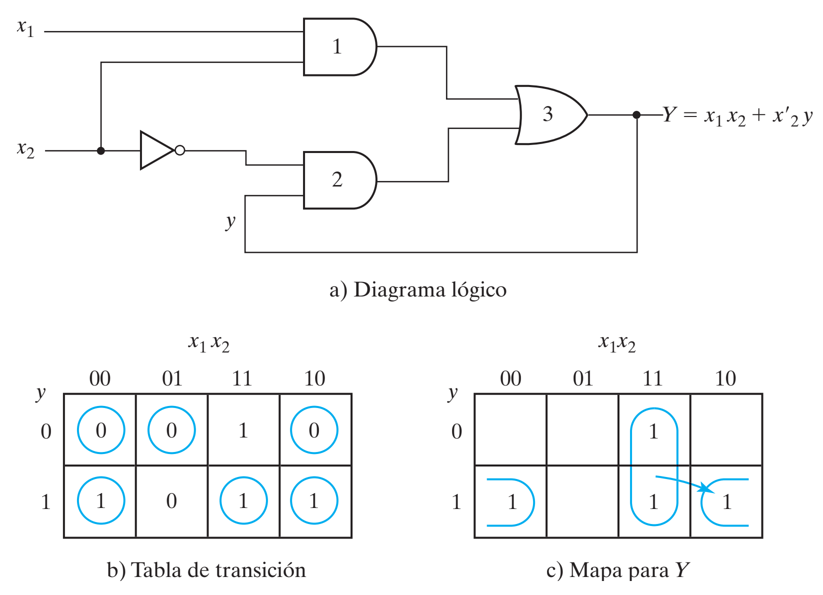


FIGURA 9-37
Peligro en un circuito secuencial asincrónico

salida se mantendrá en 1. Para aclarar esto, consideremos un latch *SR* NAND con las siguientes funciones booleanas para *S* y *R*:

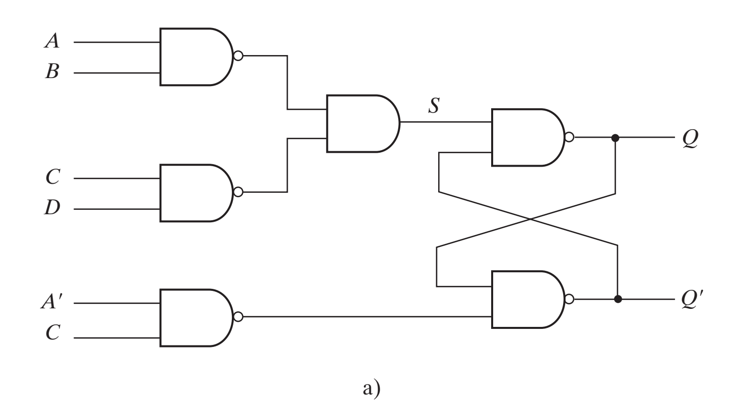
$$S = AB + CD$$
$$R = A'C$$

Puesto que se trata de un latch NAND, hay que aplicar los valores complementados a las entradas:

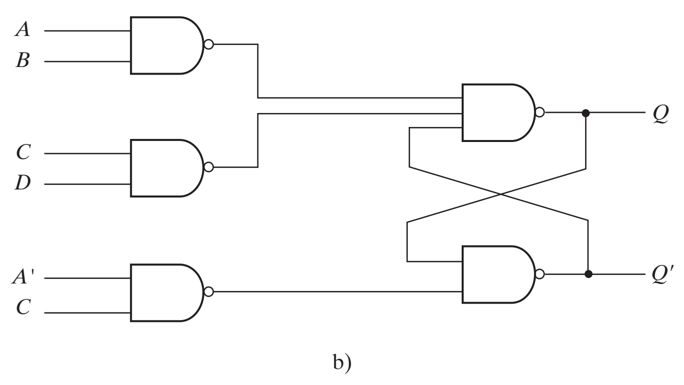
$$S = (AB + CD)' = (AB)'(CD)'$$
$$R = (A'C)'$$

Esta implementación se ilustra en la figura 9-38a). *S* se genera con dos compuertas NAND y una AND. La función booleana de la salida *Q* es

$$Q = (Q'S)' = [Q'(AB)'(CD)']'$$



a)



b)

FIGURA 9-38
Implementación con latches

Esta función se genera en la figura 9-38b) con dos niveles de compuertas NAND. Si la salida Q es 1, entonces Q' es igual a 0. Si dos de las tres entradas cambian momentáneamente a 1, la compuerta NAND asociada a la salida Q se mantendrá en 1 porque Q' se mantiene en 0.

La figura 9-38b) representa un circuito típico que puede utilizarse para construir circuitos secuenciales asincrónicos. Las dos compuertas NAND que forman el latch normalmente tienen dos entradas, pero si las funciones S o R contienen dos o más términos producto cuando se expresan en suma de productos, la compuerta NAND correspondiente del latch SR tendrá tres o más entradas. Así, los dos términos de la expresión original en suma de productos para S son AB y CD, y ambos se implementan con una compuerta NAND cuya salida se aplica a la entrada del latch NAND. De este modo, cada variable de estado requiere un circuito con dos niveles de compuertas NAND. El primer nivel consiste en compuertas NAND que implementan cada término producto de la expresión booleana original para S y R. El segundo nivel forma la conexión acoplada en cruz del latch SR con entradas que provienen de las salidas de cada compuerta NAND del primer nivel.

Peligros esenciales

Hasta aquí se han considerado los peligros caracterizados como estáticos y dinámicos. Existe otro tipo de peligros que podrían presentarse en los circuitos secuenciales asincrónicos: los llamados *peligros esenciales*. Estos peligros se deben a retardos desiguales a lo largo de dos o más trayectorias que se originan en la misma entrada. Un retardo excesivo a través de un circuito inversor, en comparación con el retardo asociado a la trayectoria de retroalimentación, podría causar un peligro así. Los peligros esenciales no se pueden corregir añadiendo compuertas redundantes, como se hace cuando hay peligros estáticos. El problema que implican se corrige ajustando el retardo en la trayectoria afectada. Para evitar peligros esenciales, cada lazo de retroalimentación se debe manejar en forma individual para asegurar que su retardo sea suficientemente largo en comparación con los retardos de otras señales que se originan en las terminales de entrada. El problema suele ser especializado, pues depende del circuito específico y de los retardos que tenga en sus diversas trayectorias.

9-8 EJEMPLO DE DISEÑO

Ya estamos en condiciones de examinar un ejemplo de diseño completo de un circuito secuencial asincrónico. Este ejemplo podría servir como referencia para el diseño de otros circuitos similares. Ilustraremos el método de diseño siguiendo los pasos del procedimiento recomendado que se numeraron al final de la sección 9-4 y que repetimos aquí. Entonces una vez establecidas las especificaciones de diseño,

1. Deduzca una tabla de flujo primitiva.
2. Reduzca la tabla de flujo fusionando filas.
3. Efectúe una asignación de estados binarios que no tenga carreras.
4. Obtenga la tabla de transición y el mapa de salida.
5. Obtenga el diagrama lógico empleando latches SR.

Especificaciones de diseño

Es necesario diseñar un flip-flop T disparado por flanco negativo. El circuito tiene dos entradas, T (*toggle*) y C (reloj), y una salida, Q. El estado de salida se complementa si $T = 1$ y el reloj C cambia de 1 a 0 (disparo por borde negativo). En todas las demás condiciones de entrada, la salida Q no cambiará. Aunque es posible usar este circuito como flip-flop en circuitos secuenciales con reloj, el diseño interno del flip-flop (como sucede con todos los demás flip-flops) es un problema asincrónico.

Tabla de flujo primitiva

La deducción de la tabla de flujo primitiva podría ser más fácil si primero obtenemos una tabla que numere todos los estados totales en que puede estar el circuito, como se aprecia en la tabla 9-6. Comenzamos con la condición de entrada $TC = 11$ y la asignamos al estado a. El circuito pasa al estado b y la salida Q se complementa de 0 a 1 cuando C cambia de 1 a 0 mientras T permanece en 1. Hay otro cambio en la salida cuando el circuito pasa del estado c al d. En este caso, $T = 1$, C cambia de 1 a 0 y la salida Q se complementa de 1 a 0. Los otros cuatro estados de la tabla no causan un cambio en la salida porque T es igual a 0. Si Q inicialmente es 0, permanece en 0, y si inicialmente es 1, permanece en 1 aunque cambie la entrada de reloj. Esta información da pie a seis estados totales. Advierta que no se han incluido las transiciones simultáneas de dos variables de entrada, como de 01 a 10, porque violan la condición de operación en modo fundamental.

La tabla de flujo primitiva se presenta en la figura 9-39. La información para esa tabla se obtiene directamente de las condiciones numeradas en la tabla 9-6. Primero se llena un cuadrado de cada fila, el correspondiente al estado estable de esa fila según se indica en la tabla. Luego se colocan guiones en los cuadrados cuya entrada difiere en dos variables de la entrada correspondiente al estado estable. A continuación se determinan las condiciones inestables con base en la información de la columna de comentarios de la tabla 9-6.

Tabla 9-6
Especificación de estados totales

	Entradas		Salida	
Estado	***T***	***C***	***Q***	**Comentarios**
a	1	1	0	Salida inicial es 0
b	1	0	1	Después del estado a
c	1	1	1	Salida inicial es 1
d	1	0	0	Después del estado c
e	0	0	0	Después del estado d o f
f	0	1	0	Después del estado e o a
g	0	0	1	Después del estado b o h
h	0	1	1	Después del estado g o c

	TC 00	01	11	10
a	–, –	f, –	(a), 0	b, –
b	g, –	–, –	c, –	(b), 1
c	–, –	h, –	(c), 1	d, –
d	e, –	–, –	a, –	(d), 0
e	(e), 0	f, –	–, –	d, –
f	e, –	(f), 0	a, –	–, –
g	(g), 1	h, –	–, –	b, –
h	g, –	(h), 1	c, –	–, –

FIGURA 9-39
Tabla de flujo primitiva

Fusión de la tabla de flujo

Las filas de la tabla de flujo primitiva se fusionan obteniendo primero todos los pares de estados compatibles. Esto se hace con ayuda de la tabla de implicación de la figura 9-40. Los cuadrados que contienen palomas definen los pares compatibles:

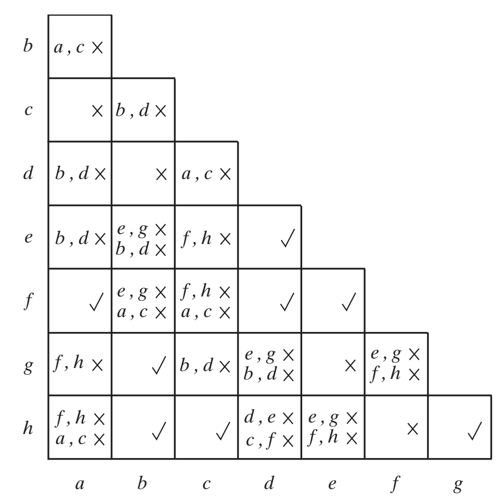
$$(a, f) \quad (b, g) \quad (b, h) \quad (c, h) \quad (d, e) \quad (d, f) \quad (e, f) \quad (g, h)$$

Los compatibles máximos se obtienen del diagrama de fusión de la figura 9-41. Los patrones geométricos que se reconocen en el diagrama son dos triángulos y dos líneas rectas. El conjuntto de máximos compatibles es

$$(a, f) \quad (b, g, h) \quad (c, h) \quad (d, e, f)$$

En este ejemplo específico, el conjunto mínimo de compatibles también es el conjunto máximo. Vemos que se satisface la condición de cerradura porque el conjunto incluye los ocho estados originales numerados en la tabla de flujo primitiva, aunque los estados h y f se repiten. La condición de cobertura también se satisface porque ningún par compatible tiene estados implicados, como se observa en la tabla de implicación.

La tabla de flujo reducida se reproduce en la figura 9-42. La de la parte a) de la figura conserva los símbolos originales de los estados pero fusiona las filas correspondientes. Por ejemplo, los estados a y f son compatibles y se fusionan en una fila que conserva las letras originales de los estados. Asimismo, se usan los otros tres conjuntos compatibles de estados para fusionar la tabla de flujo en cuatro filas, conservando los ochos símbolos de letra originales. La otra forma de dibujar la tabla de flujo fusionada se presenta en la parte b) de la figura. Ahí asigna-



	a	b	c	d	e	f	g
b	a , c ×						
c	×	b , d ×					
d	b , d ×	×	a , c ×				
e	b , d ×	e , g × b , d ×	f , h ×	√			
f	√	e , g × a , c ×	f , h × a , c ×	√	√		
g	f , h ×	√	b , d ×	e , g × b , d ×	×	e , g × f , h ×	
h	f , h × a , c ×	√	√	d , e × c , f ×	e , g × f , h ×	×	√

FIGURA 9-40
Tabla de implicación

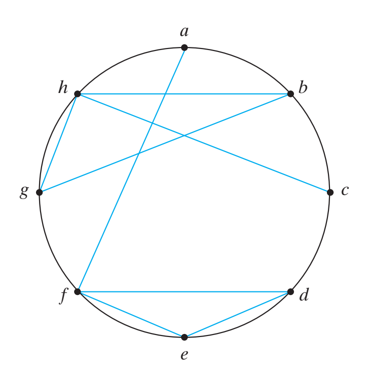


FIGURA 9-41
Diagrama de fusión

a)

	TC 00	01	11	10
a, f	e , –	(f), 0	(a), 0	b , –
b, g, h	(g), 1	(h), 1	c , –	(b), 1
c, h	g , 1	(h), 1	(c), 1	d , –
d, e, f	(e), 0	(f), 0	a , –	(d), 0

b)

	TC 00	01	11	10
a	d , –	(a), 0	(a), 0	(b), –
b	(b), 1	(b), 1	c , –	(b), 1
c	b , –	(c), 1	(c), 1	d , –
d	(d), 0	(d), 0	a , –	(d), 0

FIGURA 9-42
Tabla de flujo reducida

mos una misma letra a todos los estados estables de cada fila fusionada. Así, el símbolo *f* se sustituye por *a*, y *g* y *h* se sustituyen por *b*, y algo análogo se hace en las otras dos filas. La segunda alternativa muestra claramente una tabla de flujo de cuatro estados en la que se usan sólo cuatro símbolos de letra para los estados.

Asignación de estados y tabla de transición

El siguiente paso del diseño consiste en encontrar una asignación binaria sin carreras para los cuatro estados estables de la tabla de flujo reducida. Para encontrar una asignación adyacente adecuada, dibujamos el diagrama de transición, que corresponde a la figura 9-43. En este ejemplo es posible obtener una asignación adyacente adecuada sin necesidad de estados extra. Ello se debe a que no hay líneas diagonales en el diagrama de transición.

Al sustituir en la tabla de flujo reducida la asignación binaria indicada por el diagrama de transición, obtenemos la tabla de transición de la figura 9-44. El mapa de salida se obtiene de la tabla de flujo reducida. Asignamos valores a los guiones de la sección de salida según las reglas establecidas en la sección 9-4.

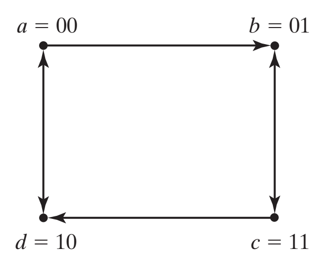


FIGURA 9-43
Diagrama de transición

a) Tabla de transición

$y_1 y_2$ \ TC	00	01	11	10
a = 00	10	(00)	(00)	01
b = 01	(01)	(01)	11	(01)
c = 11	01	(11)	(11)	10
d = 10	(10)	(10)	00	(10)

b) Mapa de salida $Q = y_2$

$y_1 y_2$ \ TC	00	01	11	10
00	0	0	0	X
01	1	1	1	1
11	1	1	1	X
10	0	0	0	0

FIGURA 9-44
Tabla de transición y mapa de salida

Diagrama lógico

El circuito a diseñar tiene dos variables de estado, Y_1 y Y_2, y una salida, Q. El mapa de salida de la figura 9-44 indica que Q es igual a la variable de estado y_2. La implementación del circuito requiere dos latches *SR*, uno para cada variable de estado. Los mapas para las entradas *S* y *R* de los dos latches se aprecian en la figura 9-45. Los mapas se obtienen a partir de la información dada en la tabla de transición utilizando las condiciones especificadas en la tabla de excitación del latch que aparece en la figura 9-14b). Las funciones booleanas simplificadas se dan abajo de cada mapa.

El diagrama lógico del circuito se observa en la figura 9-46. Aquí usamos dos latches NAND con dos o tres entradas en cada compuerta. Esta implementación sigue el patrón establecido en la sección 9-7, figura 9-38b). Las funciones de entrada *S* y *R* requieren seis compuertas NAND para su implementación.

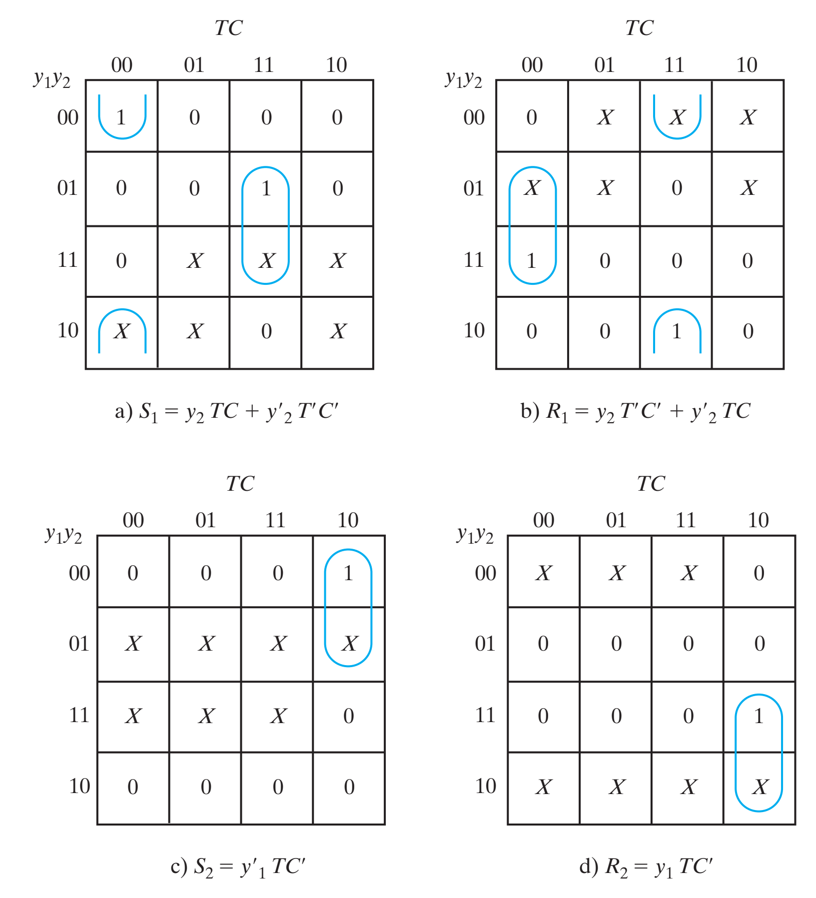


FIGURA 9-45
Mapas para las entradas de latch

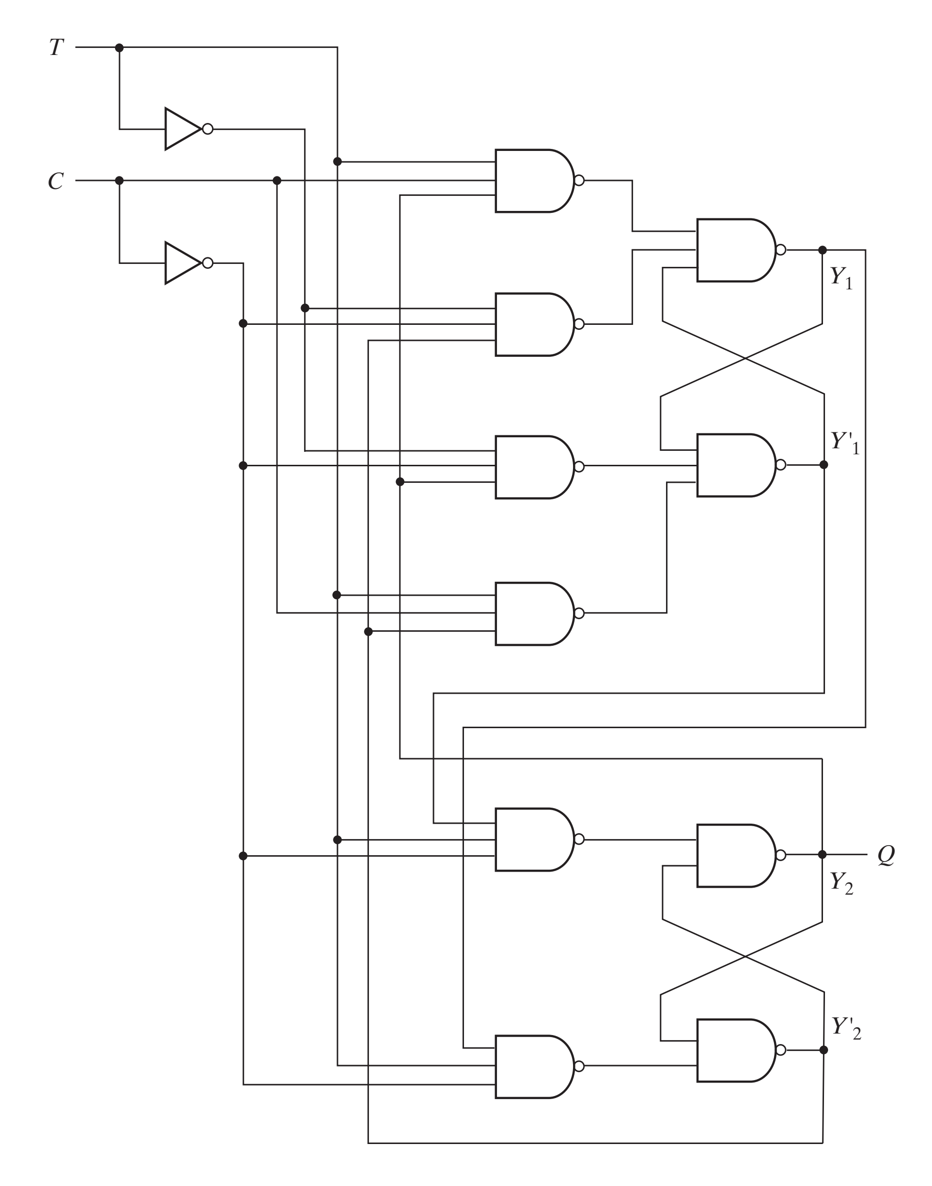


FIGURA 9-46
Diagrama lógico del flip-flop T disparado por flanco negativo

Este ejemplo ilustra lo complejo que resulta diseñar circuitos secuenciales asincrónicos. Fue necesario preparar 10 diagramas para obtener el diagrama de circuito final. Aunque la mayoría de los circuitos digitales es del tipo sincrónico, hay ocasiones en que es preciso manejar un comportamiento asincrónico. Las propiedades básicas presentadas en este capítulo son indispensables para entender plenamente el comportamiento interno de los circuitos digitales.

PROBLEMAS

9-1 a) Explique la diferencia entre los circuitos secuenciales sincrónicos y asincrónicos.

b) Defina el funcionamiento en modo fundamental.

c) Explique la diferencia entre estados estables e inestables.

d) ¿En qué difiere un estado interno de un estado total?

9-2 Deduzca la tabla de transición del circuito secuencial asincrónico de la figura P9-2. Determine la sucesión de estados internos Y_1Y_2 para esta sucesión de entradas x_1x_2: 00, 10, 11, 01, 11, 10, 00.

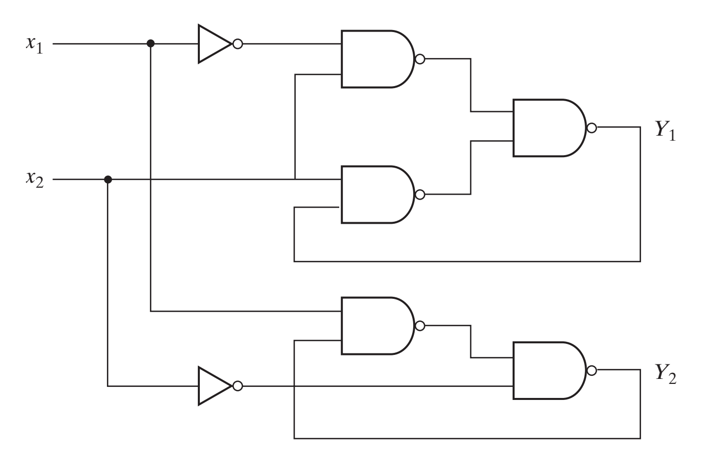


FIGURA P9-2

9-3 Un circuito secuencial asincrónico se describe con las funciones de excitación y de salida

$$Y = x_1x_2' + (x_1 + x_2')y$$
$$z = y$$

a) Dibuje el diagrama lógico del circuito.

b) Deduzca la tabla de transición y el mapa de salida.

c) Obtenga una tabla de flujo de dos estados.

d) Describa con palabras el comportamiento del circuito.

9-4 Un circuito secuencial asincrónico tiene dos estados internos y una salida. Las funciones de excitación y de salida que describen el circuito son

$$Y_1 = x_1x_2 + x_1y_2' + x_2'y_1$$
$$Y_2 = x_2 + x_1y_1'y_2 + x_1'y_1$$
$$z = x_2 + y_1$$

a) Dibuje el diagrama lógico del circuito.

b) Deduzca la tabla de transición y el mapa de salida.

c) Obtenga una tabla de flujo para el circuito.

9-5 Convierta la tabla de flujo de la figura P9-5 en una tabla de transición asignando los valores binarios siguientes a los estados: $a = 00$, $b = 11$ y $c = 01$.

a) Asigne valores al cuarto estado extra a modo de evitar carreras críticas.

b) Asigne salidas a los estados indiferentes para evitar salidas falsas momentáneas.

c) Deduzca el diagrama lógico del circuito.

	x_1x_2 00	01	11	10
a	(a), 0	b, –	c, –	(a), 1
b	a, –	(b), 0	(b), 0	c, –
c	a, –	b, –	(c), 1	(c), 0

FIGURA P9-5

9-6 Investigue la tabla de transición de la figura P9-6 y determine todas las condiciones de carrera, indicando si son o no críticas. Determine también si hay o no ciclos.

y_1y_2 \ x_1x_2	00	01	11	10
00	10	(00)	11	10
01	(01)	00	10	10
11	01	00	(11)	(11)
10	11	00	(10)	(10)

FIGURA P9-6

9-7 Analice el latch *SR* con control de la figura 5-5. Prepare la tabla de transición y demuestre que el circuito es inestable cuando las tres entradas son 1.

9-8 Modifique el diagrama de la figura 5-5a) para convertirlo en un latch tipo *JK* insertando dos conexiones de retroalimentación de las salidas a las entradas. Demuestre que el circuito es inestable cuando $J = K = 1$ mientras la entrada de control C permanece en el estado 1.

9-9 Para el circuito secuencial asincrónico de la figura P9-9,

a) Deduzca las funciones booleanas para las salidas de los dos latches *SR* Y_1 y Y_2. Observe que la entrada S del segundo latch es $x_1'y_1'$.

b) Deduzca la tabla de transición y el mapa de salida del circuito.

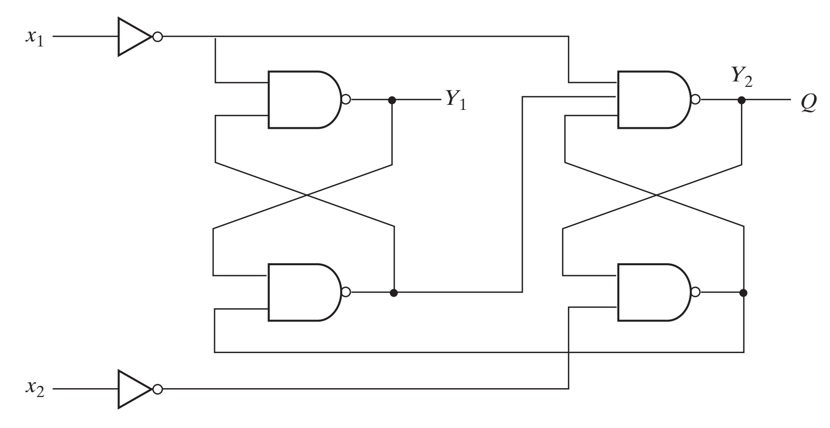


FIGURA P9-9

9-10 Implemente el circuito definido en el problema 9-3 con un latch *SR* NOR. Repita la implementación con un latch *SR* NAND.

9-11 Implemente el circuito definido en el problema 9-4 con latches *SR* NAND.

9-12 Prepare una tabla de flujo primitiva para un circuito con dos entradas, x_1 y x_2, y dos salidas, z_1 y z_2, que satisfacen estas cuatro condiciones:

a) Cuando $x_1x_2 = 00$, la salida es $z_1z_2 = 00$.

b) Cuando $x_1 = 1$ y x_2 cambia de 0 a 1, la salida es $z_1z_2 = 01$.

c) Cuando $x_2 = 1$ y x_1 cambia de 0 a 1, la salida es $z_1z_2 = 10$.

d) En los demás casos, la salida no cambia.

9-13 Se instala un semáforo en el cruce de una vía férrea y una carretera. El semáforo se controla con dos interruptores en los rieles, instalados a una distancia de 1.6 km a ambos lados del cruce. Se cierra un interruptor cuando el tren pasa sobre él, permaneciendo abierto el resto del tiempo. El semáforo cambia de verde (0 lógico) a rojo (1 lógico) cuando el frente del tren está a 1.6 km del cruce. El semáforo cambia a verde otra vez cuando la cola del tren está a 1.6 km del cruce. Suponga que la longitud del tren es menor que 3.2 km.

a) Prepare una tabla de flujo primitiva para el circuito.

b) Demuestre que la tabla de flujo se puede reducir a cuatro filas.

9-14 Es necesario diseñar un circuito secuencial asincrónico con dos entradas, x_1 y x_2, y una salida, z. Inicialmente, las dos entradas y la salida son 0. Cuando x_1 o x_2 cambia a 1, z cambia a 1. Cuando la otra entrada también cambia a 1, la salida cambia a 0. La salida permanece en 0 hasta que el circuito regresa al estado inicial.

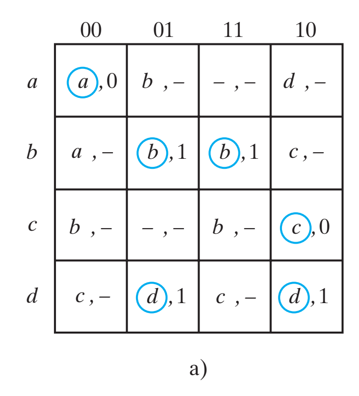
a) Prepare una tabla de flujo primitiva para el circuito y demuestre que se puede reducir a la tabla de flujo que aparece en la figura P9-14.

b) Complete el diseño del circuito.

	00	01	11	10
a	*a*, 0	*a*, 1	*b*, –	*a*, 1
b	*a*, –	*b*, 0	*b*, 0	*b*, 0

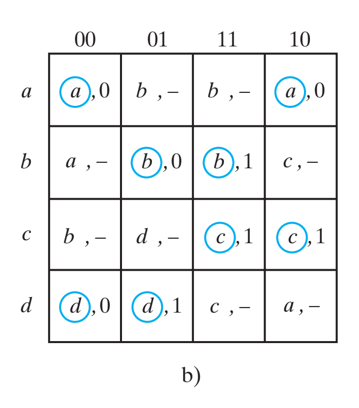
FIGURA P9-14

9-15 Asigne valores de salida a los estados indiferentes de las tablas de flujo de la figura P9-15 de forma tal que no haya pulsos transitorios en la salida.



	00	01	11	10
a	*a*, 0	*b*, –	–, –	*d*, –
b	*a*, –	*b*, 1	*b*, 1	*c*, –
c	*b*, –	–, –	*b*, –	*c*, 0
d	*c*, –	*d*, 1	*c*, –	*d*, 1

a)



	00	01	11	10
a	*a*, 0	*b*, –	*b*, –	*a*, 0
b	*a*, –	*b*, 0	*b*, 1	*c*, –
c	*b*, –	*d*, –	*c*, 1	*c*, 1
d	*d*, 0	*d*, 1	*c*, –	*a*, –

b)

FIGURA P9-15

9-16 Utilizando el método de tabla de implicación, demuestre que la tabla de estados de la tabla 5-7 no se puede reducir más.

9-17 Reduzca el número de estados de la tabla de estados dada para el problema 5-12. Utilice una tabla de implicación.

9-18 Fusione cada una de las dos tablas de flujo primitivas que se muestran en la figura P9-18. Proceda como sigue:

a) Encuentre todos los pares compatibles con la ayuda de una tabla de implicación.

b) Encuentre los máximos compatibles con la ayuda de un diagrama de fusión.

c) Encuentre un conjunto mínimo de compatibles que cubra todos los estados y sea cerrado.

	00	01	11	10
a	(*a*), 0	*b* , –	– , –	*e* , –
b	*a* , –	(*b*), 0	*c* , –	– , –
c	– , –	*d* , –	(*c*), 0	*h* , –
d	*a* , –	(*d*), 1	– , –	– , –
e	*a* , –	– , –	*f* , –	(*e*), 0
f	– , –	*g* , –	(*f*), 0	*h* , –
g	*a* , –	(*g*), 0	– , –	– , –
h	*a* , –	– , –	– , –	(*h*), 0

a)

	00	01	11	10
a	(*a*), 1	*f* , –	– , –	*e* , –
b	*c* , –	– , –	*j* , –	(*b*), 0
c	(*c*), 0	*d* , –	– , –	*b* , –
d	*c* , –	(*d*), 0	*g* , –	– , –
e	*a* , –	– , –	*g* , –	(*e*), 1
f	*a* , –	(*f*), 1	*g* , –	– , –
g	– , –	*d* , –	(*g*), 0	*k* , –
h	(*h*), 0	*d* , –	– , –	*k* , –
j	– , –	*f* , –	(*j*), 1	*b* , –
k	*a* , –	– , –	*j* , 1	(*k*), 0

b)

FIGURA P9-18

9-19 a) Encuentre una asignación binaria de estados para la tabla de flujo reducida de la figura P9-19. Evite condiciones de carrera crítica.

b) Prepare el diagrama lógico del circuito empleando latches y compuertas NAND.

x_1x_2

	00	01	11	10
a	(*a*), 0	(*a*), 1	*b* , –	*d* , –
b	*a* , –	(*b*), 0	(*b*), 0	*c* , –
c	*a* , –	– , –	*d* , –	(*c*), 0
d	*a* , –	*a* , –	(*d*), 1	(*d*), 1

FIGURA P9-19

9-20 Encuentre una asignación de estados que no tenga carreras críticas para la tabla de flujo reducida de la figura P9-20.

	00	01	11	10
a	(*a*)	*d*	(*a*)	*c*
b	*a*	(*b*)	(*b*)	*d*
c	*d*	(*c*)	*b*	(*c*)
d	(*d*)	(*d*)	*e*	(*d*)
e	*f*	*c*	(*e*)	*c*
f	(*f*)	*b*	*a*	(*f*)

FIGURA P9-20

9-21 Considere la tabla de flujo reducida de la figura P9-21.

a) Prepare el diagrama de transición y demuestre que se necesitan tres variables de estado para poder asignar estados binarios sin carreras.

b) Prepare la tabla de flujo expandida utilizando el método de asignación de múltiples filas especificado en la figura 9-32a).

	00	01	11	10
a	(*a*)	*c*	(*a*)	*d*
b	*a*	(*b*)	*c*	(*b*)
c	(*c*)	(*c*)	(*c*)	*d*
d	(*d*)	*b*	*a*	(*d*)

FIGURA P9-21

9-22 Encuentre un circuito que no tenga peligros estáticos e implemente la función booleana

$$F(A, B, C, D) = \sum(0, 2, 6, 7, 8, 10, 12)$$

9-23 Dibuje el diagrama lógico de la expresión en producto de sumas:

$$Y = (x_1 + x_2')(x_2 + x_3)$$

Demuestre que hay un peligro de 0 estático cuando x_1 y x_3 son 0 y x_2 cambia de 0 a 1. Encuentre una forma de eliminar el peligro agregando otra compuerta OR.

9-24 Las funciones booleanas para las entradas de un latch *SR* son

$$S = x_1'x_2'x_3 + x_1x_2x_3$$
$$R = x_1x_2' + x_2x_3'$$

Prepare el diagrama del circuito utilizando el mínimo de compuertas NAND.

9-25 Complete el diseño del circuito especificado en el problema 9-13.

REFERENCIAS

1. Hill, F. J. y G. R. Peterson. 1981. *Introduction to Switching Theory and Logical Design*, 3a. ed. Nueva York: John Wiley.
2. Kohavi, Z. 1978. *Switching and Automata Theory*, 2a. ed. Nueva York: McGraw-Hill.
3. McCluskey, E. J. 1986. *Logic Design Principles*. Englewood Cliffs, NJ: Prentice-Hall.
4. Unger, S. H. 1969. *Asynchronous Sequential Switching Circuits*. Nueva York: John Wiley.
5. Breeding, K. J. 1989. *Digital Design Fundamentals*. Englewood Cliffs, NJ: Prentice-Hall.
6. Friedman, A. D. 1986. *Fundamentals of Logic Design and Switching Theory*. Rockville, MD: Computer Science Press.
7. Nelson, V. P., H. T. Nagle, J. D. Irwin y B. D. Carroll 1995. *Digital Logic Circuit Analysis and Design*. Prentice-Hall.

10 Circuitos integrados digitales

10-1 INTRODUCCIÓN

Los circuitos integrados (CI) y las familias de lógica digital se estudiaron en la sección 2-8. En este capítulo se presentarán los circuitos electrónicos de cada familia de lógica digital en CI y se analizará su funcionamiento eléctrico. Suponemos que el lector tiene conocimientos básicos de circuitos eléctricos.

Las familias de lógica digital en CI que se examinarán aquí son:

RTL	Lógica resistor-transistor
DTL	Lógica diodo-transistor
TTL	Lógica transistor-transistor
ECL	Lógica acoplada por emisor
MOS	Metal-óxido-semiconductor
CMOS	Metal-óxido-semiconductor complementario

Las primeras dos, RTL y DTL, sólo tienen importancia histórica, porque ya no se usan en el diseño de sistemas digitales. RTL fue la primera familia comercial que se usó ampliamente. La incluimos aquí porque sirve como punto de partida para explicar el funcionamiento básico de las compuertas digitales. Los circuitos DTL han sido reemplazados por TTL. De hecho, TTL es una modificación de la compuerta DTL. Será más fácil entender el funcionamiento de la compuerta TTL una vez que hayamos analizado la compuerta DTL. TTL, ECL y CMOS tienen un gran número de circuitos SSI, además de componentes MSI, LSI y VLSI.

El circuito básico de cada familia de lógica digital en CI es una compuerta NAND o NOR. Este circuito básico es el bloque de construcción primario del que se derivan todos los demás componentes digitales, más complejos. Cada familia de lógica en CI tiene un libro de datos que numera todos los circuitos integrados de la familia. Las diferencias en las funciones lógicas que cada familia ofrece no radican tanto en las funciones que desempeñan como

Entradas		Salida
x	y	z
L	L	H
L	H	H
H	L	H
H	H	L

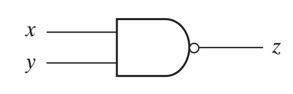


FIGURA 10-1
Compuerta NAND de lógica positiva

en las características eléctricas específicas de la compuerta básica con la que se construye el circuito.

Las compuertas NAND y NOR suelen definirse por las funciones booleanas que implementan en términos de variables binarias. Al analizarlas como circuitos electrónicos, es necesario investigar sus relaciones de entrada-salida en términos de dos niveles de voltaje: un nivel *alto* denotado por H (*high*) y un nivel *bajo* denotado por L (*low*). Como se mencionó en la sección 2-8, la asignación de 1 binario a H da pie a un sistema de lógica positiva, y la asignación de 1 binario a L da pie a un sistema de lógica negativa. En la figura 10-1 se presenta la tabla de verdad de una compuerta NAND de lógica positiva en términos de H y L. Vemos que la salida de la compuerta es alta en tanto una o más entradas sean bajas. La salida es baja sólo cuando ambas entradas son altas. El comportamiento de una compuerta NAND de lógica positiva en términos de señales altas y bajas se plantea así:

Si *cualquier* entrada de una compuerta NAND es baja, la salida es alta.

Si *todas* las entradas de una compuerta NAND son altas, la salida es baja.

En la figura 10-2 se presenta la tabla de verdad de una compuerta NOR de lógica positiva. La salida de esta compuerta es baja cuando una o más entradas son altas. La salida es alta cuando ambas entradas son bajas. El comportamiento de una compuerta NOR de lógica positiva en términos de señales altas y bajas se describe así:

Si *cualquier* entrada de una compuerta NOR es alta, la salida es baja.

Si *todas* las entradas de una compuerta NOR son bajas, la salida es alta.

Es preciso recordar estas características de las compuertas NAND y NOR porque se usarán en el análisis de las compuertas electrónicas en este capítulo.

Entradas		Salida
x	y	z
L	L	H
L	H	L
H	L	L
H	H	L

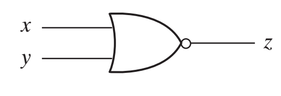


FIGURA 10-2
Compuerta NOR de lógica positiva

Un transistor de unión bipolar (BJT, *bipolar junction transistor*) puede ser *npn* o *pnp*. En contraste, los transistores de efecto de campo (FET, *field-effect transistors*) son unipolares. El funcionamiento de un transistor bipolar depende del flujo de dos tipos de portadores de carga: electrones y huecos. Un transistor unipolar sólo depende del flujo de un tipo de portador mayoritario, que puede consistir en electrones (canal *n*) o huecos (canal *p*). Las primeras cuatro familias de lógica digital de nuestra lista —RTL, DTL, TTL y ECL— usan transistores bipolares. Las últimas dos familias —MOS y CMOS— usan un tipo de transistor unipolar llamado transistor de efecto de campo de metal-óxido-semiconductor (abreviado MOSFET o simplemente MOS).

En este capítulo, primero se hablará de las características más comunes que se usan para comparar las familias de lógica digital. Luego se describirán las propiedades del transistor bipolar y se analizarán las compuertas básicas de las familias de lógica bipolar. Después, se explicará el funcionamiento del transistor MOS y presentaremos las compuertas básicas de sus dos familias de lógica.

10-2 CARACTERÍSTICAS ESPECIALES

Las características de las familias de lógica digital de CI suelen compararse analizando el circuito de la compuerta básica de cada familia. Los parámetros más importantes que se evalúan y comparan son abanico de salida (*fan out*), disipación de potencia, retardo de propagación y margen de ruido. Primero explicaremos las propiedades de estos parámetros y luego los usaremos para comparar las familias de lógica en CI.

Abanico de salida

El abanico de salida (*fan-out* o carga máxima de salida) de una compuerta especifica el número de cargas estándar que es posible conectar a la salida de la compuerta sin degradar su funcionamiento normal. Por lo regular, una carga estándar se define como la cantidad de corriente que requiere una entrada de otra compuerta de la misma familia de lógica. A veces se usa el término *carga* en lugar de abanico de salida. Este término proviene del hecho de que la salida de una compuerta es capaz de suministrar una cantidad limitada de corriente, más allá de la cual deja de operar correctamente y se dice que está “sobrecargada”. La salida de una compuerta por lo regular se conecta a las entradas de otras compuertas. Cada entrada consume cierta cantidad de corriente de la salida de la compuerta, de modo que cada conexión adicional aumenta la carga sobre la compuerta. A veces se especifican reglas de carga para una familia de circuitos digitales. Estas reglas especifican la carga máxima permitida para cada salida de cada circuito de la familia. Si se excede la carga máxima especificada podría haber un funcionamiento incorrecto porque el circuito no puede suministrar la potencia que se le exige. El abanico de salida es el número máximo de entradas que es posible conectar a la salida de una compuerta, y se expresa con un número.

El abanico de salida se calcula a partir de la cantidad de corriente que está disponible en la salida de una compuerta y la cantidad de corriente que cada entrada de una compuerta necesita. Consideremos las conexiones que se muestran en la figura 10-3. La salida de una compuerta se conecta a una o más entradas de otras compuertas. En la figura 10-3a), la salida de la compuerta está en el nivel de alto voltaje; alimenta una corriente I_{OH} a todas las entradas de compuerta conectadas a ella. Cada entrada de compuerta requiere una corriente I_{IH} para operar correctamente. En la figura 10-3b), la salida de la compuerta está en el nivel de bajo voltaje.

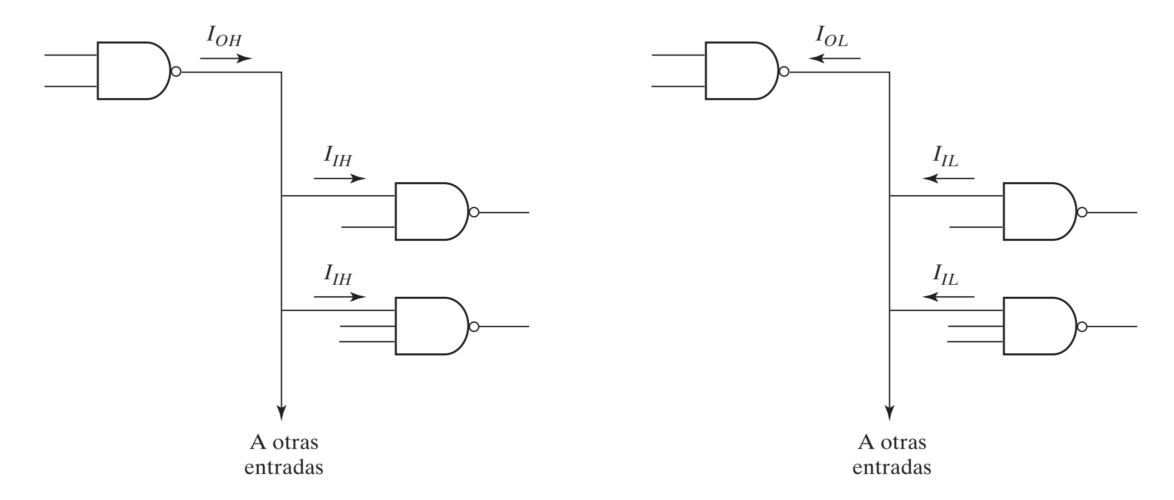


FIGURA 10-3
Cálculo de abanico de salida

Proporciona un drenaje de corriente I_{OL} a todas las entradas de compuerta conectadas a ella. Cada entrada de compuerta alimenta una corriente I_{IL}. El abanico de salida de la compuerta se calcula a partir del cociente I_{OH}/I_{IH} o I_{OL}/I_{IL}, lo que sea menor. Por ejemplo, las compuertas TTL estándar tienen estos valores para las corrientes:

$$I_{OH} = 400\ \mu\text{A}$$
$$I_{IH} = 40\ \mu\text{A}$$
$$I_{OL} = 16\ \text{mA}$$
$$I_{IL} = 1.6\ \text{mA}$$

En este caso, los dos cocientes dan el mismo número:

$$\frac{400\ \mu\text{A}}{40\ \mu\text{A}} = \frac{16\ \text{mA}}{1.6\ \text{mA}} = 10$$

Por tanto, el abanico de salida de la TTL estándar es 10. Esto implica que la salida de una compuerta TTL no se puede conectar a más de 10 entradas de otras compuertas de la misma familia de lógica. De lo contrario, la compuerta podría ser incapaz de alimentar o drenar la cantidad de corriente que necesitan las entradas conectadas a ella.

Disipación de potencia

Todo circuito electrónico requiere cierta cantidad de potencia eléctrica para operar. La disipación de potencia es un parámetro que se expresa en miliwatts (mW) y representa la cantidad de potencia que la compuerta necesita. Este parámetro no incluye la potencia suministrada por otra compuerta; más bien, representa la potencia suministrada a la compuerta por la fuente de poder. Un CI con cuatro compuertas requerirá de su fuente de poder cuatro veces la potencia disipada en cada compuerta.

La cantidad de potencia disipada en una compuerta se calcula con base en el voltaje de alimentación V_{CC} y la corriente I_{CC} que el circuito consume. La potencia es el producto $V_{CC} \times I_{CC}$. El consumo de corriente de la fuente de poder depende del estado lógico de la compuerta. La corriente consumida cuando la salida de la compuerta está en el nivel de voltaje alto se denomina I_{CCH}. Cuando la salida está en el nivel de voltaje bajo, la corriente es I_{CCL}. La corriente media es

$$I_{CC}(\text{avg}) = \frac{I_{CCH} + I_{CCL}}{2}$$

y sirve para calcular la disipación media de potencia:

$$P_D(\text{avg}) = I_{CC}(\text{avg}) \times V_{CC}$$

Por ejemplo, una compuerta NAND TTL estándar utiliza un voltaje de alimentación V_{CC} de 5V y tiene consumos de corriente de $I_{CCH} = 1$ mA e $I_{CCL} = 3$ mA. La corriente media es $(3 + 1)/2 = 2$ mA. La disipación media de potencia es $5 \times 2 = 10$ mW. Un CI con cuatro compuertas NAND disipa un total de $10 \times 4 = 40$ mW. En un sistema digital típico hay muchos CI, y es preciso considerar la potencia que cada uno requiere. La disipación total de potencia del sistema es la sumatoria de las potencias disipadas en todos los CI.

Retardo de propagación

El retardo de propagación de una compuerta es el tiempo medio de transición que la señal tarda en propagarse de la entrada a la salida cuando la señal binaria cambia de valor. Las señales que pasan por una compuerta tardan cierto tiempo en propagarse desde las entradas hasta la salida. Este lapso se define como retardo de propagación de la compuerta y se mide en nanosegundos (ns). 1 ns es igual a 10^{-9} segundos.

Las señales que viajan de las entradas de un circuito digital a sus salidas pasan por una serie de compuertas. La sumatoria de los retardos de propagación a través de las compuertas es el retardo total del circuito. Si la rapidez de operación es importante, cada compuerta debe tener un retardo de propagación corto, y el circuito digital debe tener el mínimo de compuertas entre las entradas y las salidas.

El retardo de propagación medio de una compuerta se calcula a partir de las formas de onda de entrada y salida, como se indica en la figura 10-4. El tiempo de retardo de la señal entre la entrada y la salida cuando la salida cambia del nivel alto al bajo se denota con t_{PHL}. Asimismo, cuando la salida pasa del nivel bajo al alto, el retardo es t_{PLH}. Se acostumbra medir el tiempo entre el punto de 50% en las transiciones de entrada y de salida. En general, los dos retardos son desiguales, y ambos varían dependiendo de las condiciones de carga. La duración media del retardo de propagación se calcula promediando los dos retardos.

Por ejemplo, los retardos de una compuerta TTL estándar son $t_{PHL} = 7$ ns y $t_{PLH} = 11$ ns. Estas cantidades se dan en el libro de datos de TTL y se miden con una resistencia de carga de 400 ohms y una capacitancia de carga de 15 pF. El retardo de propagación medio de la compuerta TTL es $(11 + 7)/2 = 9$ ns.

Bajo ciertas condiciones, es más importante conocer el máximo tiempo de retardo de una compuerta que el valor promedio. El libro de datos TTL lista los siguientes retardos de propagación máximos para una compuerta NAND estándar: $t_{PHL} = 15$ ns y $t_{PHL} = 22$ ns. Cuando la velocidad de operación es crítica, es necesario tomar en cuenta el retardo máximo para garantizar una operación adecuada.

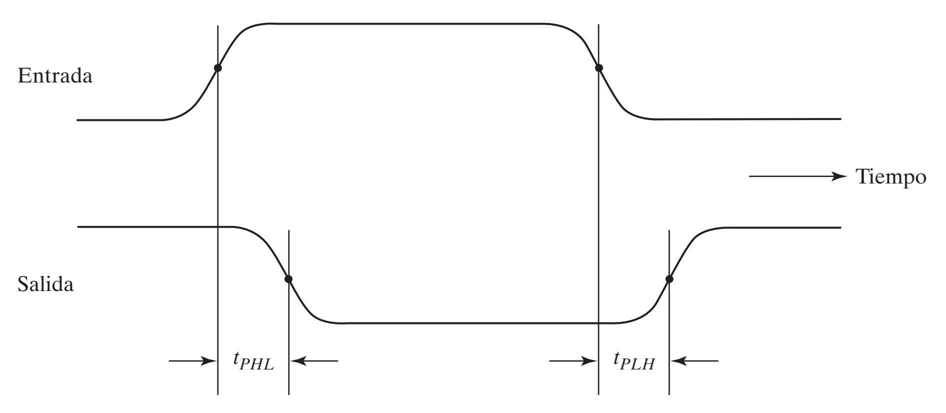


FIGURA 10-4
Medición del retardo de propagación

Las señales de entrada en la mayoría de los circuitos digitales se aplican simultáneamente a más de una compuerta. Todas las compuertas conectadas a entradas externas constituyen el primer nivel lógico del circuito. Las compuertas que reciben al menos una entrada proveniente de una salida de una compuerta de primer nivel se consideran parte del segundo nivel lógico, y así para el tercer y demás niveles lógicos superiores. El retardo de propagación total del circuito es igual al retardo de propagación de una compuerta multiplicado por el número de niveles lógicos del circuito. De esta forma, una reducción en el número de niveles lógicos disminuye el retardo de propagación y hace que el circuito sea más rápido. La reducción del retardo de propagación de los circuitos podría ser más importante que la reducción del número total de compuertas, si la rapidez de operación es un factor principal.

Margen de ruido

Las señales eléctricas espurias de fuentes industriales y de otro tipo en ocasiones inducen voltajes indeseables en los alambres que conectan circuitos lógicos. Esas señales indeseables se denominan *ruido*. Hay dos tipos de ruido que considerar. El ruido de corriente continua (c.c.) se debe a una deriva en los niveles de voltaje de una señal. El ruido de corriente alterna (c.a.) es un pulso aleatorio que podría tener su origen en otras señales de conmutación. Así pues, “ruido” es un término empleado para referirse a una señal indeseable superpuesta a la señal operativa normal. El *margen de ruido* es el voltaje máximo de ruido añadido a una señal de entrada de un circuito digital que no causa un cambio indeseable en la salida del circuito. La capacidad de los circuitos para operar de manera confiable en un entorno ruidoso es importante en muchas aplicaciones. El margen de ruido se expresa en volts y representa la señal máxima de ruido que la compuerta puede tolerar.

El margen de ruido se calcula a partir del voltaje de señal disponible en la salida de la compuerta y el voltaje de señal requerido en la entrada de la compuerta. La figura 10-5 ilustra las señales que se usan para calcular el margen de ruido. La parte a) muestra el intervalo de voltajes de salida que pueden presentarse en una compuerta típica. Cualquier voltaje en la salida de la compuerta entre V_{CC} y V_{OH} se considera como estado de nivel alto, y cualquier voltaje entre 0 y V_{OL} en la salida de la compuerta se considera como estado de nivel bajo. Los voltajes entre V_{OL} y V_{OH} son indeterminados y no se presentan en condiciones operativas normales como no sea durante la transición entre los dos niveles. Los dos intervalos de voltaje correspondientes que la entrada de la compuerta reconoce se indican en la figura 10-5b). Para compensar cualquier señal de rui-

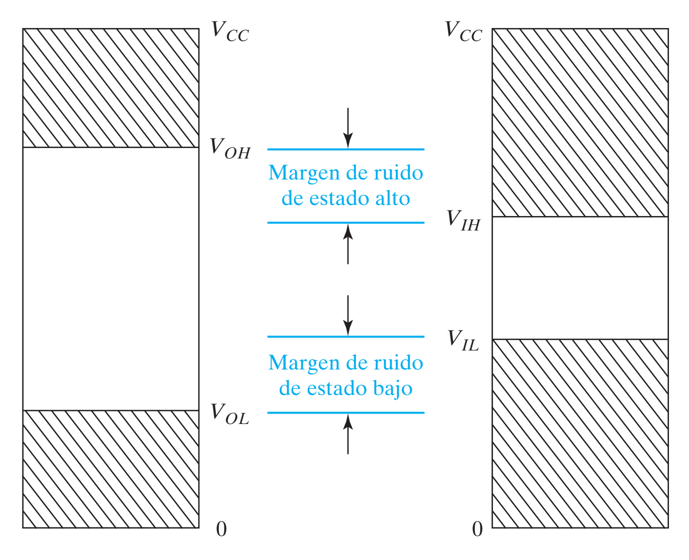


FIGURA 10-5
Señales para evaluar el margen de ruido

do, el circuito debe diseñarse de modo que V_{IL} sea mayor que V_{OL} y V_{IH} sea menor que V_{OH}. El margen de ruido es la diferencia $V_{OH} - V_{IH}$ o $V_{IL} - V_{OL}$, lo que sea menor.

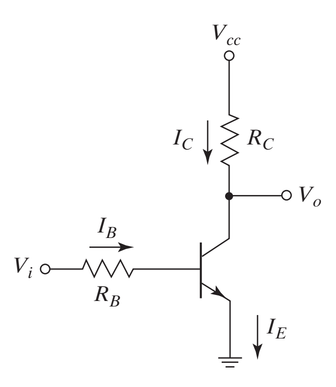
Como se ilustra en la figura 10-5, V_{OL} es el voltaje máximo que puede tener la salida en el estado de nivel bajo. El circuito tolera cualquier señal de ruido menor que el margen de ruido ($V_{IL} - V_{OL}$) porque la entrada reconoce la señal como nivel bajo. Cualquier señal mayor que V_{OL} más el valor del margen de ruido hará que el voltaje de entrada llegue al intervalo indeterminado, lo cual podría causar un error en la salida de la compuerta. De forma similar, un ruido de voltaje negativo mayor que $V_{OH} - V_{IH}$ pondrá el voltaje de entrada dentro del intervalo indeterminado.

Los parámetros para el margen de ruido en una compuerta NAND TTL estándar son $V_{OH} = 2.4$ V, $V_{OL} = 0.4$ V, $V_{IH} = 2$ V y $V_{IL} = 0.8$ V. El margen de ruido en el estado alto es $2.4 - 2 = 0.4$ V, y en el estado bajo, $0.8 - 0.4 = 0.4$ V. En este caso, ambos valores son iguales.

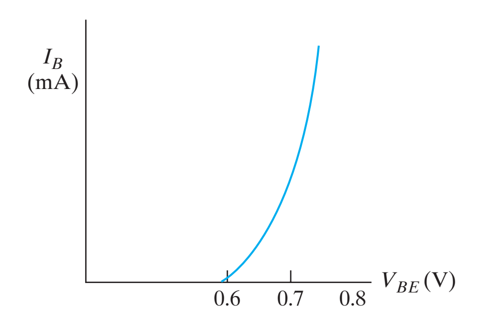
10-3 CARACTERÍSTICAS DE TRANSISTOR BIPOLAR

Esta sección hará un repaso del transistor bipolar aplicado a circuitos digitales. Usaremos esta información para analizar el circuito básico de las cuatro familias de lógica bipolar. Los transistores bipolares pueden ser del tipo *npn* o *pnp*, y se construyen con material semiconductor de germanio o silicio. Los transistores en CI, empero, se hacen con silicio y suelen ser del tipo *npn*.

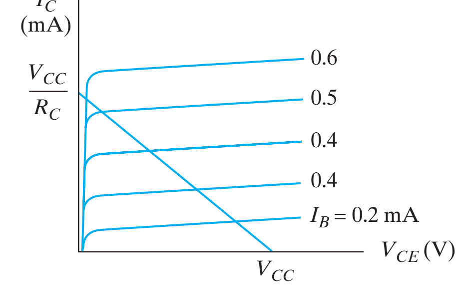
Los datos básicos que se necesitan para analizar circuitos digitales se obtienen de las curvas de características típicas de un transistor de silicio *npn* de emisor común, las cuales se muestran en la figura 10-6. El circuito de a) es un inversor simple con dos resistores y un tran-



a) Circuito inversor



b) Característica de la base del transistor



c) Característica del colector del transistor

FIGURA 10-6
Características del transistor *npn* de silicio

sistor. La corriente marcada I_C fluye a través del resistor R_C y el colector del transistor. La corriente I_B fluye a través del resistor R_B y la base del transistor. El emisor está conectado a tierra y su corriente es $I_E = I_C + I_B$. El voltaje de alimentación está entre V_{CC} y tierra. La entrada está entre V_i y tierra, y la salida, entre V_o y tierra.

Hemos supuesto una dirección positiva para las corrientes, como se indica. Éstas son las direcciones en que fluyen normalmente las corrientes en un transistor *npn*. Las corrientes de colector y de base, I_C e I_B, respectivamente, son positivas cuando entran en el transistor. La corriente de emisor, I_E, es positiva cuando sale del transistor, como indica la flecha en la terminal del emisor. El símbolo V_{CE} representa la caída de potencial del colector al emisor y siempre es positiva. Asimismo, V_{BE} es la caída de potencial en la unión base-emisor. Esta unión tiene polarización directa cuando V_{BE} es positiva, y polarización inversa cuando V_{BE} es negativa.

La característica gráfica base-emisor se señala en la figura 10-6b). Se trata de una gráfica de V_{BE} contra I_B. Si el voltaje base-emisor es menor que 0.6 V, se dice que el transistor está *en corte* y no fluye corriente de base. Cuando la unión base-emisor tiene polarización directa con un voltaje de más de 0.6 V, el transistor conduce e I_B comienza a subir con gran rapidez, mientras que V_{BE} casi no cambia. El voltaje V_{BE} por un transistor conductor raras veces excede 0.8 V.

Las características gráficas de colector-emisor, junto con la línea de carga, se aprecian en la figura 10-6c). Cuando V_{BE} es menor que 0.6 V, el transistor queda en corte con $I_B = 0$, y la corriente que fluye en el colector es insignificante. Entonces, el circuito colector-emisor se comporta como un circuito abierto. En la región *activa*, el voltaje del colector, V_{CE}, puede tener cualquier valor entre aproximadamente 0.8 V y V_{CC}. Se calcula que la corriente de colector I_C en esta región es aproximadamente igual a $h_{FE}I_B$, donde h_{FE} es un parámetro del transistor llamado *ganancia de corriente c.c.* La corriente máxima del colector no depende de I_B, sino del circuito externo conectado al colector. Ello se debe a que V_{CE} siempre es positivo y su valor más bajo posible es 0 V. Por ejemplo, en el inversor que se muestra, la I_C máxima se obtiene haciendo $V_{CE} = 0$ para obtener $I_C = V_{CC}/R_C$.

Ya dijimos que $I_C = h_{FE}I_B$ en la región activa. El parámetro h_{FE} varía ampliamente dentro del intervalo operativo del transistor, pero aun así resulta útil utilizar un valor medio para fines de análisis. En un intervalo operativo típico, h_{FE} es aproximadamente 50, pero en ciertas condiciones podría bajar hasta 20. Cabe señalar que la corriente de base I_B podría aumentarse hasta cualquier valor deseado, pero la corriente de colector I_C está limitada por parámetros del circuito externo. Por ello, podría darse una situación en la que $h_{FE}I_B$ sea mayor que I_C. En un caso así, se dice que el transistor está en la región de *saturación*. Por tanto, la condición de saturación está dada por la relación

$$I_B \geq \frac{I_{CS}}{h_{FE}}$$

donde I_{CS} es la corriente de colector máxima que fluye durante la saturación. V_{CE} no es exactamente cero en la región de saturación, pero normalmente anda cerca de 0.2 V.

Los datos básicos que se necesitan para analizar circuitos digitales con transistores bipolares se dan en la tabla 10-1. En la región de corte, V_{BE} es menor que 0.6 V, V_{CE} se considera como circuito abierto, y ambas corrientes son insignificantes. En la región activa, V_{BE} es de aproximadamente 0.7 V, V_{CE} podría variar en un intervalo amplio e I_C se calcula en función de I_B. En la región de saturación, V_{BE} casi no cambia, pero V_{CE} baja a 0.2 V. La corriente de base debe ser lo bastante alta como para satisfacer la desigualdad señalada. Para simplificar el análisis, supondremos que $V_{BE} = 0.7$ V si el transistor está conduciendo, sea en la región activa o en la de saturación.

Podemos analizar circuitos digitales utilizando el siguiente procedimiento recomendado: para cada transistor del circuito, determine si su V_{BE} es menor que 0.6 V. Si lo es, el transistor está en corte y el circuito de colector a emisor se considera un circuito abierto. Si V_{BE} es mayor que 0.6 V, el transistor podría estar en la región activa o en la de saturación. Se calcula la corriente de base, suponiendo $V_{BE} = 0.7$ V. Luego se calcula el valor máximo posible de corriente de colector, I_{CS}, suponiendo $V_{CE} = 0.2$ V. Estos cálculos se harán en términos de voltajes apli-

Tabla 10-1
Parámetros típicos de transistor npn de silicio

Región	V_{BE} (V)	V_{CE} (V)	Relación de corriente
En corte	< 0.6	Circuito abierto	$I_B = I_C = 0$
Activa	0.6–0.7	> 0.8	$I_C = h_{FE}I_B$
Saturación	0.7–0.8	0.2	$I_B \geq I_{CS}/h_{FE}$

cados y valores de resistores. Luego, si la corriente de base es lo bastante grande como para que $I_B \geq I_{CS}/h_{FE}$, se deduce que el transistor está en la región de saturación con $V_{CE} = 0.2$ V. Por otra parte, si la corriente de base es más pequeña y no satisface la relación anterior, el transistor estará en la región activa y se recalculará la corriente de colector I_C con la ecuación $I_C = h_{FE}I_B$.

Como ejemplo, consideremos el circuito inversor de la figura 10-6a) con los parámetros siguientes:

$$R_C = 1\ k\Omega \qquad V_{CC} = 5\text{ V (fuente de voltaje)}$$

$$R_B = 22\text{ k}\Omega \qquad H = 5\text{ V (voltaje de alto nivel)}$$

$$h_{FE} = 50 \qquad L = 0.2\text{ V (voltaje de bajo nivel)}$$

Con un voltaje de entrada $V_i = L = 0.2$ V, tenemos que $V_{BE} < 0.6$ V y el transistor está en corte. El circuito colector-emisor se comporta como circuito abierto, así que el voltaje de salida $V_o = 5\text{ V} = H$.

Con un voltaje de entrada $V_i = H = 5$ V, deducimos que $V_{BE} > 0.6$ V. Suponiendo que $V_{BE} = 0.7$, calculamos la corriente de base:

$$I_B = \frac{V_i - V_{BE}}{R_B} = \frac{5 - 0.7}{22\text{ k}\Omega} = 0.195\text{ mA}$$

La corriente máxima de colector, suponiendo $V_{CE} = 0.2$ V, es

$$I_{CS} = \frac{V_{CC} - V_{CE}}{R_C} = \frac{5 - 0.2}{1\text{ k}\Omega} = 4.8\text{ mA}$$

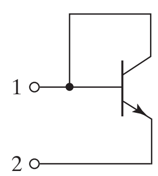
Luego vemos si hay saturación, empleando la condición

$$0.195 = I_B \geq \frac{I_{CS}}{h_{FE}} = \frac{4.8}{50} = 0.096\text{ mA}$$

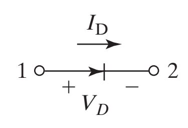
y vemos que la desigualdad se satisface, porque $0.195 > 0.096$. Concluimos que el transistor está saturado y el voltaje de salida $V_o = V_{CE} = 0.2\text{ V} = L$. Así pues, el circuito se comporta como inversor.

El procedimiento recién descrito se utilizará mucho al analizar circuitos en las secciones que siguen. Esto se hará con un análisis cualitativo, es decir, sin escribir las ecuaciones numéricas específicas. El análisis cuantitativo y los cálculos específicos se dejarán como ejercicios en la sección de problemas al final del capítulo.

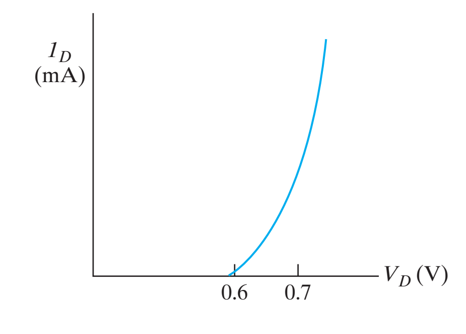
Hay ocasiones en las que se usan no sólo transistores sino también diodos en los circuitos digitales. Un diodo en CI por lo regular se construye a partir de un transistor con su colector conectado a la base, como se observa en la figura 10-7a). En la figura 10-7b) se incluye el símbolo gráfico que denota un diodo. El diodo se comporta en esencia como la unión base-emisor de un transistor. Su característica gráfica, que se muestra en la figura 10-7c), es similar a la característica base-emisor de un transistor. Podemos concluir, entonces, que un diodo está apagado y no conduce cuando su voltaje directo, V_D, es menor que 0.6 V. Cuando el diodo conduce, la corriente I_D fluye en la dirección que se indica en la figura 10-7b) y V_D permanece cerca de 0.7 V. Siempre hay que incluir un resistor externo para limitar la corriente en un diodo conductor, porque su voltaje permanece relativamente constante y es de una fracción de volt.



a) Transistor adaptado para usarse como diodo



b) Símbolo gráfico de diodo



c) Característica de diodo

FIGURA 10-7
Símbolo y característica del diodo de silicio

10-4 CIRCUITOS RTL Y DTL

Compuerta RTL básica

El circuito básico de la familia de lógica digital RTL es la compuerta NOR que se observa en la figura 10-8. Cada entrada está asociada a un resistor y un transistor. Los colectores de los transistores están conectados en la salida. Los niveles de voltaje para el circuito son 0.2 V para el nivel bajo y de 1 a 3.6 V para el nivel alto.

El análisis de la compuerta RTL es muy sencillo y sigue el procedimiento delineado en la sección anterior. Si cualquier entrada de la compuerta RTL es alta, el transistor correspondiente se lleva a saturación. Esto hace que la salida sea baja, sin importar los estados de los otros transistores. Si todas las entradas están bajas, en 0.2 V, todos los transistores estarán en corte

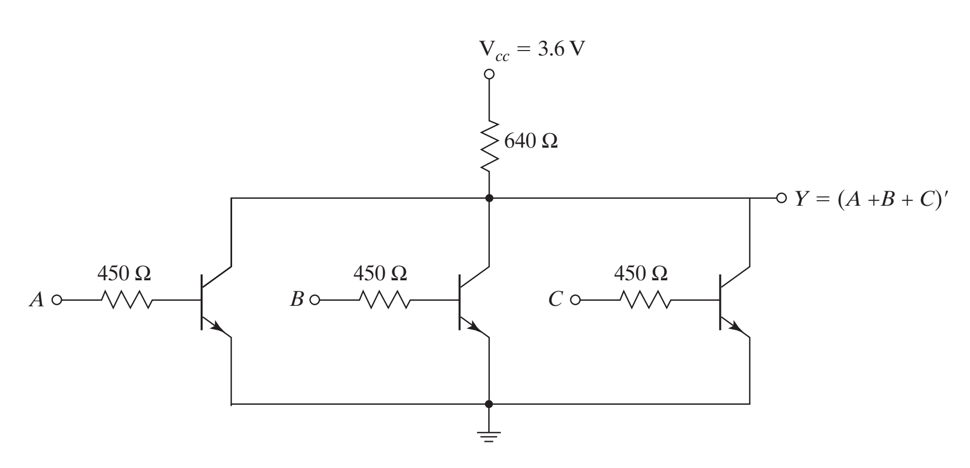


FIGURA 10-8
Compuerta NOR RTL básica

porque $V_{BE} < 0.6$ V. Esto hace que la salida del circuito sea alta y se aproxime al valor del voltaje de alimentación V_{CC}. Esto confirma las condiciones planteadas en la figura 10-2 para la compuerta NOR. El margen de ruido para una entrada de señal baja es $0.6 - 0.2 = 0.4$ V.

El abanico de salida de la compuerta RTL está limitado por el valor del voltaje de salida cuando es alto. Al cargarse la salida con las entradas de otras compuertas, la carga consume más corriente. Esta corriente debe fluir a través del resistor de 640 Ω. Un sencillo cálculo (véase el problema 10-2) demuestra que, si h_{FE} baja a 20, el voltaje de salida baja a aproximadamente 1 V cuando el abanico de salida es 5. Cualquier voltaje por debajo de 1 V en la salida podría no llevar al siguiente transistor a la saturación, como se requiere. La disipación de potencia de la compuerta RTL es de aproximadamente 12 mW y el retardo de propagación promedia 25 ns.

Compuertas básicas DTL

El circuito básico de la familia de lógica digital DTL es la compuerta NAND que se aprecia en la figura 10-9. Cada entrada está asociada a un diodo. Los diodos y el resistor de 5 kΩ forman una compuerta AND. El transistor actúa como amplificador de corriente al tiempo que invierte la señal digital. Los dos niveles de voltaje son 0.2 V para el nivel bajo y entre 4 y 5 V para el nivel alto.

El análisis de la compuerta DTL debe ajustarse a las condiciones numeradas en la figura 10-1 para la compuerta NAND. Si cualquier entrada de la compuerta está baja, en 0.2 V, el diodo de entrada correspondiente conduce corriente a través de V_{CC} y el resistor de 5 kΩ al nodo de entrada. El voltaje en el punto P es igual al voltaje de entrada de 0.2 V más una caída de diodo de 0.7 V, dando un total de 0.9 V. Para que el transistor comience a conducir, es preciso que el voltaje en el punto P supere un potencial de una caída de V_{BE} en $Q1$ más dos caídas de diodo en $D1$ y $D2$, o sea, $3 \times 0.6 = 1.8$ V. Puesto que el diodo conductor de entrada mantiene en 0.9 V el voltaje en P, el transistor queda en corte y el voltaje de salida es alto, 5 V.

Si todas las entradas de la compuerta son altas, el transistor se empuja a la región de saturación. El voltaje en P ahora es igual a V_{BE} más las dos caídas de diodo en $D1$ y $D2$, o sea, $0.7 \times 3 = 2.1$ V. Puesto que todas las entradas son altas en 5 V, y $V_P = 2.1$ V, los diodos de entrada están polarizados a la inversa y, por tanto, apagados. La corriente de base es igual a la diferencia de las corrientes que fluyen por los dos resistores de 5 kΩ y es suficiente para lle-

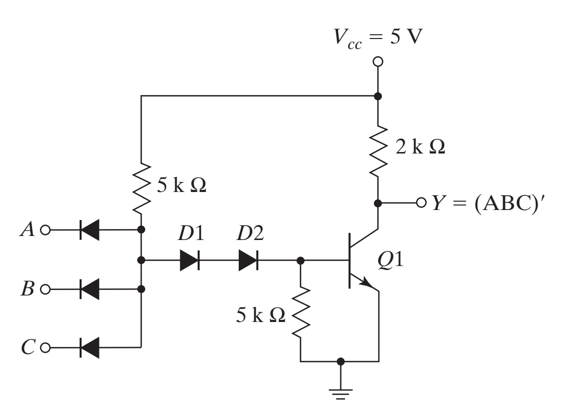


FIGURA 10-9
Compuerta NAND DTL básica

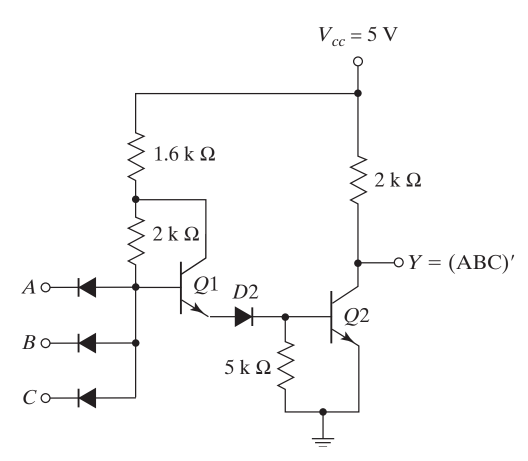


FIGURA 10-10
Compuerta DTL modificada

var el transistor a la saturación. (Véase el problema 10-3.) Con el transistor saturado, la salida cae a V_{CE}, que es 0.2 V, el nivel bajo de la compuerta.

La disipación de potencia de una compuerta DTL es de aproximadamente 12 mW y el retardo de propagación promedia 30 ns. El margen de ruido es de aproximadamente 1 V y puede lograrse un abanico de salida de hasta 8. El abanico de salida de la compuerta DTL está limitado por la corriente máxima que puede fluir en el colector del transistor saturado. (Véase el problema 10-4.)

El abanico de salida de una compuerta DTL aumenta sustituyendo uno de los diodos del circuito de base por un transistor, como se observa en la figura 10-10. El transistor $Q1$ se mantiene en la región activa cuando el transistor de salida $Q2$ está saturado. En consecuencia, el circuito modificado puede alimentar una mayor cantidad de corriente de base al transistor de salida, el cual ahora puede consumir una mayor cantidad de corriente de colector antes de quedar fuera de saturación. Una parte de la corriente de colector proviene de los diodos conductores en las compuertas de carga cuando $Q2$ está saturado. Así, un incremento en la corriente saturada de colector aceptable permite conectar más cargas a la salida, lo que aumenta el abanico de salida de la compuerta.

10-5 LÓGICA DE TRANSISTOR-TRANSISTOR (TTL)

La compuerta TTL básica original era una pequeña mejora a la compuerta DTL. Al progresar la tecnología TTL, se añadieron otras mejoras que hicieron que esta familia de lógica se usara ampliamente en el diseño de sistemas digitales. Hay varias subfamilias o series de tecnología TTL. Los nombres y características de las ocho series TTL se dan en la tabla 10-2. Los CI TTL comerciales tienen una designación numérica que comienza con 74 seguido de un sufijo que identifica el tipo de serie. Como ejemplos podemos citar los 7404, 74S86 y 74ALS161. El abanico de salida, la disipación de potencia y el retardo de propagación se definieron en la sección 10-2. El producto rapidez-potencia es un parámetro importante para comparar las diversas series TTL: es el producto del retardo de propagación y la disipación de potencia y se mide en picojoules (pJ). Es deseable que este parámetro tenga un valor bajo, porque indica que es posible lograr un retardo de propagación dado sin una disipación de potencia excesiva, y viceversa.

Tabla 10-2
Series TTL y sus características

Nombre de serie TTL	Prefijo	Abanico de salida	Disipación de potencia (mW)	Retardo de propagación (ns)	Producto rapidez-potencia (pJ)
Estándar	74	10	10	9	90
Baja potencia	74L	20	1	33	33
Alta velocidad	74H	10	22	6	132
Schottky	74S	10	19	3	57
Schottky de baja potencia	74LS	20	2	9.5	19
Schottky avanzado	74AS	40	10	1.5	15
Schottky avanzado de baja potencia	74ALS	20	1	4	4
Rápido	74F	20	4	3	12

La compuerta TTL estándar fue la primera versión de la familia TTL. Luego, esta compuerta básica se diseñó con diferentes valores de resistor para producir compuertas con menor disipación de potencia o más rápidas. El retardo de propagación de un circuito de transistor que entra en saturación depende principalmente de dos factores: tiempo de almacenamiento y constantes de tiempo *RC*. Si se reduce el tiempo de almacenamiento disminuye el retardo de propagación. Si se reducen los valores de los resistores en el circuito se reducen las constantes de tiempo *RC* y disminuye el retardo de propagación. Desde luego, el precio es una mayor disipación de potencia porque las resistencias más bajas extraen más corriente de la fuente de poder. La rapidez de la compuerta es inversamente proporcional al retardo de propagación.

En la compuerta TTL de baja potencia, los valores de los resistores son más altos que en la compuerta estándar, a fin de reducir la disipación de potencia, pero el retardo de propagación es mayor. En la compuerta TTL de alta velocidad, se reducen los valores de los resistores para acortar el retardo de propagación, pero aumenta la disipación de potencia. La compuerta TTL Schottky fue la siguiente mejora en la tecnología. El efecto del transistor Schottky es eliminar el retardo de tiempo de almacenamiento impidiendo al transistor entrar en saturación. Esta serie aumenta su rapidez de operación sin un aumento excesivo en la disipación de potencia. El TTL Schottky de baja potencia sacrifica algo de rapidez a cambio de una menor disipación de potencia. Tiene el mismo retardo de propagación que el TTL estándar, pero su disipación de potencia es apenas la quinta parte. Innovaciones recientes han dado pie al desarrollo de la serie Schottky avanzada. Tiene un menor retardo de propagación que la serie Schottky y su disipación de potencia también es menor. El Schottky avanzado de baja potencia es el que más bajo producto rapidez-potencia tiene y es la serie más eficiente. La familia TTL rápida es la mejor opción para diseños de alta velocidad.

Todas las series TTL están disponibles en SSI y en formas más complejas como componentes MSI y LSI. Las diferencias entre las series TTL no radican en la lógica digital que ejecutan, sino en la construcción interna de la compuerta NAND básica. De cualquier modo, las compuertas TTL de todas las series disponibles pueden tener tres tipos de configuración de salida:

1. Salida de colector abierto
2. Salida en *totem pole*
3. Salida de tres estados

Consideraremos estos tres tipos de salidas al describir el circuito de la compuerta TTL básica.

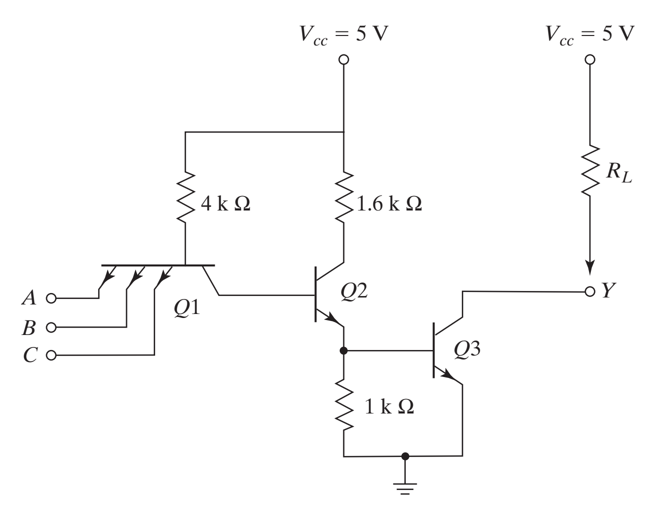


FIGURA 10-11
Compuerta TTL de colector abierto

Compuerta de salida de colector abierto

La compuerta TTL básica que se ilustra en la figura 10-11 es un circuito de compuerta DTL modificado. Los múltiples emisores del transistor *Q*1 están conectados a las entradas. La mayor parte del tiempo, estos emisores se comportan como los diodos de entrada de la compuerta DTL, ya que forman una unión *pn* con su base común. La unión base-colector de *Q*1 actúa como otro diodo de unión *pn* correspondiente a *D*1 en la compuerta DTL (véase la figura 10-5). El transistor *Q*2 sustituye al segundo diodo, *D*2, de la compuerta DTL. La salida de la compuerta TTL se toma del colector abierto de *Q*3. Es preciso insertar externamente al paquete de CI un resistor conectado a V_{CC} para que la salida "suba" (*pull up*) al nivel alto de voltaje cuando *Q*3 está apagado; en caso contrario, la salida actúa como circuito abierto. Más adelante se explicará por qué el resistor no debe ser interno.

Los dos niveles de voltaje de la compuerta TTL son 0.2 V para el nivel bajo y de 2.4 a 5 V para el nivel alto. El circuito básico es una compuerta NAND. Si cualquier entrada es baja, la unión base-emisor correspondiente en *Q*1 estará polarizada en directo. El voltaje en la base de *Q*1 es igual al voltaje de salida de 0.2 V más una caída V_{BE} de 0.7 o 0.9 V. Para que *Q*3 comience a conducir, la trayectoria de *Q*1 a *Q*3 debe vencer un potencial de una caída de diodo en la unión *pn* base-colector de *Q*1 y dos caídas V_{BE} en *Q*2 y *Q*3, o sea, $3 \times 0.6 = 1.8$ V. Puesto que la señal de entrada mantiene a la base de *Q*1 en 0.9 V, el transistor de salida no puede conducir y queda en corte. El nivel de salida será alto si se conecta un resistor externo entre la salida y V_{CC} (o circuito abierto si no se usa un resistor).

Si todas las entradas son altas, tanto *Q*2 como *Q*3 conducirán y se saturarán. El voltaje de base de *Q*1 es igual al voltaje en su unión *pn* base-colector más dos caídas V_{BE} en *Q*2 y *Q*3, o sea, aproximadamente $0.7 \times 3 = 2.1$ V. Puesto que todas las entradas son altas y mayores que 2.4 V, todas las uniones base-emisor de *Q*1 estarán polarizadas en reversa. Cuando el transistor de salida *Q*3 se satura (a condición de que tenga una trayectoria de corriente), el voltaje de salida pasará al nivel bajo, 0.2 V. Esto confirma las condiciones para una operación NAND.

En este análisis, dijimos que la unión base-colector de *Q*1 actúa como un diodo de unión *pn*. Esto es verdad en la condición de estado estable. Sin embargo, durante la transición de apagado, *Q*1 sí exhibe una acción de transistor, la cual reduce el retardo de propagación. Cuando todas las salidas son altas y luego se lleva una de las entradas a un nivel bajo, tanto *Q*2 como *Q*3 comienzan a apagarse. En este momento, la unión de colector de *Q*1 estará polarizada en reversa y el emisor tendrá polarización directa; por tanto, el transistor *Q*1 entrará momentáneamente en la región activa. La corriente de colector de *Q*1 proviene de la base de *Q*2 y rápidamente elimina la carga sobrante almacenada en *Q*2 durante su estado de saturación anterior. Esto causa una reducción en el tiempo de almacenamiento del circuito, en comparación con el tipo de entrada DTL. El resultado es una disminución del tiempo de apagado de la compuerta.

La compuerta TTL de colector abierto opera sin el resistor externo cuando está conectada a entradas de otras compuertas TTL, aunque esto no es recomendable debido a la baja inmunidad al ruido que se observa. Sin un resistor externo, la salida de la compuerta será un circuito abierto cuando *Q*3 esté apagado. Un circuito abierto en una entrada de una compuerta TTL se comporta como si tuviera una entrada de alto nivel (pero una pequeña cantidad de ruido puede convertirla en nivel bajo). Cuando *Q*3 conduce, su colector tiene una trayectoria de corriente alimentada por la entrada de la compuerta de carga a través de V_{CC}, el resistor de 4 kΩ y la unión base-emisor polarizada en directo.

Las compuertas de colector abierto se usan en tres aplicaciones principales: control de una lámpara o relevador, ejecución de lógica alambrada y construcción de un sistema de *bus* común. Una salida de colector abierto puede controlar una lámpara colocada en su salida a través de un resistor limitante. Cuando la salida es baja, el transistor saturado *Q*3 forma una trayectoria para la corriente que enciende la lámpara. Cuando el transistor de salida está apagado, la lámpara se apaga porque no hay trayectoria para la corriente.

Si se conectan las salidas de varias compuertas TTL de colector abierto con un solo resistor externo, se ejecuta una lógica de AND alambrada. Recuerde que una función AND de lógica positiva sólo produce un nivel alto si todas las variables están altas; de lo contrario, la función estará baja. Si se conectan entre sí las salidas de compuertas de colector abierto, la salida común sólo será alta cuando todos los transistores de salida estén apagados (o altos). Si un transistor de salida conduce, hace que la salida pase al estado bajo.

La lógica alambrada que se ejecuta con las compuertas TTL de colector abierto (oc, *open collector*) se ilustra en la figura 10-12. El alambrado físico en a) indica cómo deben conectar-

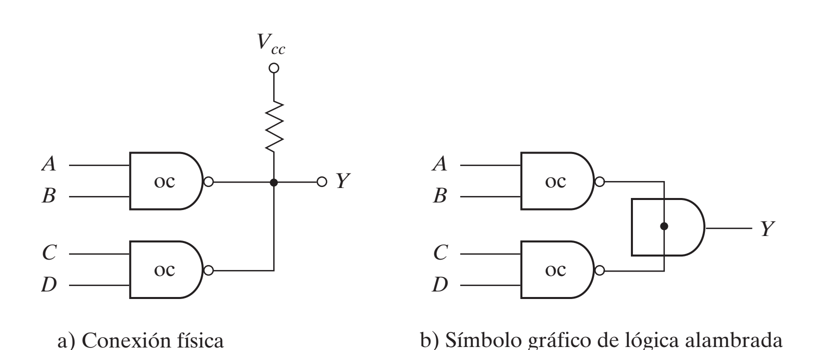


FIGURA 10-12
AND alambrada de dos compuertas de colector abierto (oc), $Y = (AB + CD)'$

se las salidas a un resistor común. El símbolo gráfico de una conexión así se muestra en b). La función AND que se forma conectando las dos salidas se denomina función AND alambrada. La compuerta AND se dibuja con líneas que pasan por el centro de la compuerta para distinguirla de una compuerta convencional. La compuerta AND alambrada no es una compuerta física; sólo es un símbolo que designa la función que se obtiene de la conexión indicada. La función booleana que se obtiene del circuito de la figura 10-12 es la operación AND entre las salidas de las dos compuertas NAND:

$$Y = (AB)' \cdot (CD)' = (AB + CD)'$$

Se prefiere la segunda expresión porque muestra una operación que se conoce comúnmente como función AND-OR-INVERT (véase la sección 3-7).

Es factible conectar las compuertas de colector abierto para formar un bus común. En cualquier instante, todas las salidas de compuerta conectadas al bus, salvo una, deben mantenerse en su estado alto. La compuerta seleccionada podría estar en el estado alto o en el bajo, dependiendo de si queremos transmitir un 1 o un 0 por el bus. Se requieren circuitos de control para seleccionar la compuerta específica que controla el bus en cualquier momento dado.

La figura 10-13 representa la conexión de cuatro fuentes a una línea de bus común. Cada una de las cuatro entradas alimenta un inversor de colector abierto, y las salidas de los inversores se juntan para formar una sola línea de bus. La figura revela que tres de las entradas son 0, lo que produce un 1 o nivel alto en el bus. La cuarta entrada, I_4, está ahora en posibilidad de transmitir información a través de la línea de bus común al inversor 5. Recuerde que en la lógica alambrada se efectúa una operación AND. Si $I_4 = 1$, la salida de la compuerta 4 será 0 y la operación AND alambrada producirá un 0. Si $I_4 = 0$, la salida de la compuerta 4 será 1 y la operación AND alambrada producirá un 1. Así, si todas las demás salidas se mantienen en 1, la compuerta seleccionada podrá transmitir su valor a través del bus. El valor transmitido es el complemento de I_4, pero el inversor 5 en el extremo receptor fácilmente puede invertir esta señal otra vez para hacer $Y = I_4$.

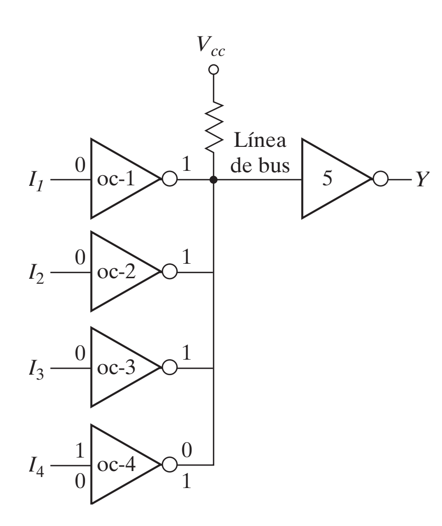


FIGURA 10-13
Compuertas de colector abierto que forman una línea de bus común

Salida en totem pole

La impedancia de salida de una compuerta normalmente es una carga resistiva más una capacitiva. La carga capacitiva consiste en la capacitancia del transistor de salida, la capacitancia de las compuertas del abanico de salida y cualquier capacitancia que pudiera tener el cableado. Cuando la salida cambia del estado bajo al alto, el transistor de salida de la compuerta cambia de saturación a corte y la capacitancia total de carga C se carga exponencialmente desde el nivel de voltaje bajo hasta el alto con una constante de tiempo igual a RC. En el caso de la compuerta de colector abierto, R es el resistor externo marcado R_L. Para un valor operativo típico de $C = 15$ pF y $R_L = 4$ kΩ, el retardo de propagación de una compuerta TTL de colector abierto durante el tiempo de apagado es de 35 ns. Si un circuito *de actuación (pull-up) activo* sustituye al resistor de actuación pasivo R_L, el retardo de propagación se reduce a 10 ns. Esta configuración, que se expone en la figura 10-14, se denomina salida en *totem pole* porque el transistor $Q4$ está "montado" sobre $Q3$.

La compuerta TTL con salida en totem pole es igual a la compuerta de colector abierto, excepto por el transistor de salida $Q4$ y el diodo $D1$. Cuando la salida Y está en el estado bajo, $Q2$ y $Q3$ se llevan a la saturación como en la compuerta de colector abierto. El voltaje en el colector de $Q2$ es $V_{BE}(Q3) + V_{CE}(Q2)$ o $0.7 + 0.2 = 0.9$ V. La salida $Y = V_{CE}(Q3) = 0.2$ V. El transistor $Q4$ está en corte porque, para que empiece a conducir, su base debe ser una caída de V_{BE} más una caída de diodo, o sea, $2 \times 0.6 = 1.2$ V. Puesto que el colector de $Q2$ está conectado a la base de $Q4$, el voltaje de este último es de sólo 0.9 V en lugar de los 1.2 V requeridos, así que $Q4$ queda en corte. Se incluye el diodo en el circuito para que haya una caída de diodo en la trayectoria de salida, lo que garantiza que $Q4$ quedará en corte cuando $Q3$ esté saturado.

Cuando la salida cambia al estado alto porque una de las entradas cae al estado bajo, los transistores $Q2$ y $Q3$ quedan en corte. Sin embargo, la salida se mantiene momentáneamente baja

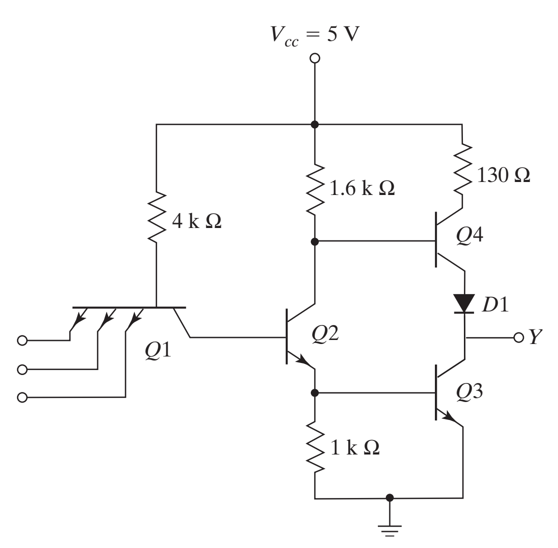


FIGURA 10-14
Compuerta TTL con salida en totem pole

porque los voltajes en la capacitancia de carga no pueden cambiar instantáneamente. Tan pronto como $Q2$ se apaga, $Q4$ conduce porque su base está conectada a V_{CC} a través del resistor de 1.6 kΩ. La corriente requerida para cargar la capacitancia de carga hace que $Q4$ se sature momentáneamente, y el voltaje de salida aumenta con una constante de tiempo RC. Sin embargo, R en este caso es 130 Ω más la resistencia de saturación de $Q4$, más la resistencia del diodo, lo que da un total de aproximadamente 150 Ω. Este valor de R es mucho menor que la resistencia de actuación pasiva que se usa en el circuito de colector abierto. Por ello, la transición del nivel bajo al alto es mucho más rápida.

Al cargarse la carga capacitiva, el voltaje de salida aumenta y la corriente $Q4$ disminuye, poniendo al transistor en la región activa. Así, en contraste con los otros transistores, $Q4$ está en la región *activa* cuando la condición es de estado estable. El valor final del voltaje de salida es entonces 5 V menos una caída de V_{BE} en $Q4$ menos una caída de diodo en $D1$, o sea, aproximadamente 3.6 V. El transistor $Q3$ entra en corte muy rápidamente, pero durante el tiempo de transición inicial tanto $Q3$ como $Q4$ están encendidos y se extrae una corriente pico de la fuente de poder. Este pico de corriente genera ruido en el sistema de distribución de la fuente de poder. Si el cambio de estado es frecuente, los picos de corriente transitoria elevan el consumo de corriente de la fuente de poder y la disipación de potencia del circuito aumenta.

La conexión de lógica alambrada no se permite con circuitos de salida en totem pole. Cuando dos salidas en totem pole se conectan y la salida de una compuerta es alta mientras que la de la otra es baja, la cantidad excesiva de corriente consumida puede generar suficiente calor como para dañar los transistores del circuito (véase el problema 10-7). Algunas compuertas TTL se construyen de modo que resistan la cantidad de corriente que fluye en estas condiciones. En todo caso, la corriente de colector en la compuerta baja podría ser suficiente para llevar el transistor a la región activa y producir en la conexión alambrada un voltaje de salida mayor que 0.8 V, lo cual no es una señal binaria válida en las compuertas TTL.

Compuerta TTL Schottky

Como ya se dijo, una reducción del tiempo de almacenamiento reduce el retardo de propagación. Ello se debe a que el tiempo necesario para que un transistor salga de la saturación retarda el cambio del transistor de la condición encendido a la condición apagado. Es posible eliminar la saturación colocando un diodo Schottky entre la base y el colector de cada transistor saturado del circuito. El diodo Schottky se forma con la unión de un metal y un semiconductor, en contraste con los diodos convencionales, que se forman con la unión de un semiconductor tipo p y uno tipo n. El voltaje en un diodo Schottky conductor es de sólo 0.4 V, en vez de los 0.7 V de un diodo convencional. La presencia de un diodo Schottky entre la base y el colector impide al transistor entrar en saturación. El transistor resultante se llama *transistor Schottky*. El uso de transistores Schottky en un TTL reduce el retardo de propagación sin que aumente la disipación de potencia.

La compuerta TTL Schottky se representa en la figura 10-15. Observe el símbolo especial que se utiliza para los transistores y diodos Schottky. El diagrama indica que todos los transistores son del tipo Schottky excepto $Q4$. Este transistor es la excepción porque no se satura, sino que se mantiene en la región activa. Advierta también que los valores de los resistores se han reducido para disminuir aún más el retardo de propagación.

Además de usar transistores Schottky y resistores de más bajo valor, el circuito de la figura 10-15 incluye otras modificaciones que no estaban presentes en la compuerta estándar de la figura 10-14. Se han añadido dos nuevos transistores, $Q5$ y $Q6$, y se han insertado diodos Schot-

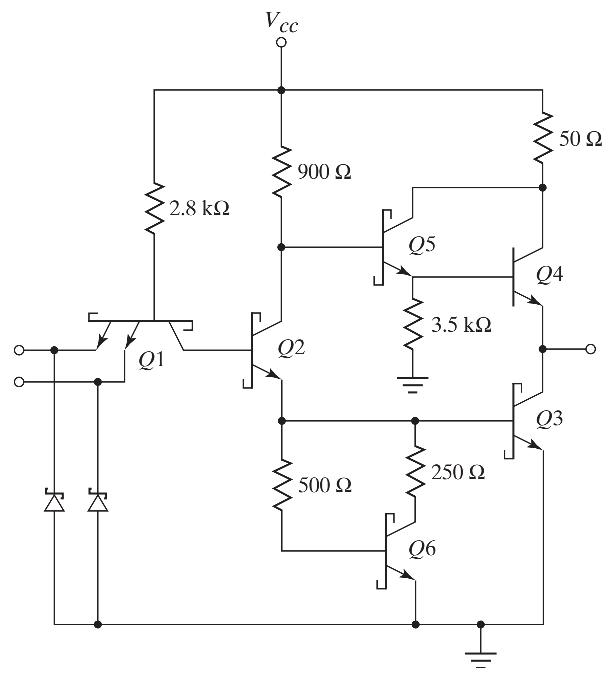


FIGURA 10-15
Compuerta TTL Schottky

tky entre cada terminal de entrada y tierra. No hay diodo en el circuito de totem pole, pero la nueva combinación de *Q*5 y *Q*4 sigue produciendo las dos caídas de V_{BE} necesarias para evitar que *Q*4 conduzca cuando la salida es baja. Esta combinación comprende un doble emisor-seguidor llamado *par Darlington*. Éste proporciona una ganancia de corriente muy alta y una resistencia extremadamente baja. Esto es exactamente lo que se necesita durante el cambio de baja a alta de la salida, y el resultado es una disminución en el retardo de propagación.

Los diodos en cada entrada que se aprecian en el circuito ayudan a amortiguar cualquier oscilación que pudiera presentarse en las líneas de entrada. En condiciones de conmutación transitoria, las líneas de señal parecen inductivas; esto, junto con una capacitancia parásita, hace que las señales oscilen. Cuando la salida de una compuerta cambia del estado bajo al alto, la forma de onda oscilante en la entrada podría tener excursiones por debajo de tierra de hasta 2-3 V, dependiendo de la longitud de la línea. Los diodos conectados a tierra amortiguan esta oscilación porque conducen tan pronto como el voltaje negativo excede 0.4 V. Al limitarse la excursión negativa, también se reduce la oscilación positiva. Estos diodos fijadores han limitado tan bien los efectos de línea, que todas las versiones de compuertas TTL los utilizan.

El resistor en el emisor de *Q*2 en la figura 10-14 se ha reemplazado en la figura 10-15 por un circuito que consiste en el transistor *Q*6 y dos resistores. El efecto de este circuito es reducir los picos de corriente de apagado antes mencionados. El análisis de este circuito, que ayuda a reducir el tiempo de propagación de la compuerta, es demasiado complicado como para presentarlo en esta breve explicación.

Compuerta de tres estados

Como ya se mencionó, las salidas de dos compuertas TTL con estructura de totem pole no pueden conectarse entre sí como en las salidas de colector abierto. No obstante, existe un tipo especial de compuerta en totem pole que permite la conexión alambrada de salidas para formar un sistema de bus común. Cuando una compuerta TTL con salida en totem pole tiene esta propiedad, se le denomina compuerta de *tres estados*.

Una compuerta de tres estados presenta tres estados de salida: 1) un estado de bajo nivel cuando el transistor inferior del totem pole está encendido y el de arriba está apagado, 2) un estado de alto nivel cuando el transistor superior del totem pole está encendido y el de abajo está apagado y 3) un tercer estado en el que ambos transistores del totem pole están apagados. El tercer estado es de circuito abierto o alta impedancia, y permite una conexión con alambre directo de muchas salidas a una línea común. Las compuertas de tres estados hacen innecesarias las compuertas de colector abierto en configuraciones de bus.

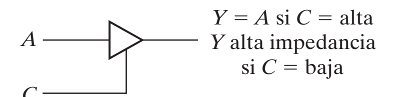
La figura 10-16a) incluye el símbolo gráfico de una compuerta búfer de tres estados. Cuando la entrada de control *C* es alta, la compuerta está habilitada y se comporta como un búfer normal cuya salida es igual al valor binario de entrada. Cuando la entrada de control es baja, la salida es un circuito abierto, lo que produce una impedancia alta (el tercer estado) independientemente del valor de la entrada *A*. Algunas compuertas de tres estados producen un estado de alta impedancia cuando la entrada de control es alta. Esto se representa simbólicamente en la figura 10-16b). Aquí tenemos dos burbujas, una para la salida del inversor y la otra para indicar que la compuerta está habilitada cuando *C* es baja.

El diagrama de circuito del inversor de tres estados se reproduce en la figura 10-16c). Los transistores *Q*6, *Q*7 y *Q*8 asociados a la entrada de control forman un circuito similar al de la compuerta de colector abierto. Los transistores *Q*1 a *Q*5, asociados a la entrada de datos, forman un circuito TTL en totem pole. Los dos circuitos se conectan entre sí a través del diodo *D*1. Al igual que en un circuito de colector abierto, el transistor *Q*8 se apaga cuando la entrada de control *C* está en el estado de nivel bajo. Esto impide al diodo *D*1 conducir. Además, el emisor de *Q*1 conectado a *Q*8 no cuenta con una trayectoria de conducción. En esta condición, el transistor *Q*8 no afecta la operación de la compuerta y la salida *Y* depende únicamente de la entrada de datos en *A*.

Cuando la entrada de control es alta, el transistor *Q*8 se enciende y la corriente que fluye desde V_{CC} a través del diodo *D*1 hace que el transistor *Q*8 se sature. El voltaje en la base de *Q*5 es ahora igual al voltaje en el transistor saturado, *Q*8, más una caída de diodo, es decir, 0.9 V. Este voltaje apaga a *Q*5 y *Q*4, porque es menor que dos caídas de V_{BE}. Al mismo tiempo, la entrada baja a uno de los emisores de *Q*1 hace que se apague el transistor *Q*3 (y *Q*2). Así, tanto *Q*3 como *Q*4 en el totem pole se apagan y la salida del circuito se comporta como circuito abierto con impedancia de salida muy alta.

Se crea un bus de tres estados alambrando entre sí varias salidas de tres estados. En cualquier instante, sólo una entrada de control está habilitada, mientras que todas las demás salidas están en el estado de alta impedancia. La compuerta que no está en estado de alta impedancia puede transmitir información binaria a través del bus común. Es crucial que todas menos una de las salidas estén en el tercer estado; de lo contrario, tendremos la condición indeseable de dos salidas en totem pole activas conectadas entre sí.

Una característica importante de casi todas las compuertas de tres estados es que el retardo de habilitación de la salida es más largo que el de inhabilitación de la salida. Si un circuito de control habilita una compuerta e inhabilita otra al mismo tiempo, la compuerta inhabilitada



a) Compuerta búfer de tres estados

A
C
Y = A′ si C = baja
Y alta impedancia
si C = alta

b) Compuerta inversora de tres estados

V_{CC}
Entrada de datos A
Entrada de control C
Q_1 Q_2 Q_3 Q_4 Q_5 Q_6 Q_7 Q_8 D_1 Y

c) Diagrama de circuito del inversor de tres estados en b)

FIGURA 10-16
Compuerta TTL de tres estados

pasa al estado de alta impedancia antes de que la otra compuerta quede habilitada. Esto elimina la posibilidad de que ambas compuertas estén activas al mismo tiempo.

Hay una corriente de fuga muy pequeña asociada a la condición de alta impedancia de una compuerta de tres estados. Sin embargo, es tan pequeña que es posible conectar hasta 100 compuertas de tres estados para formar una línea de bus común.

10-6 LÓGICA ACOPLADA POR EMISOR (ECL)

La lógica acoplada por emisor (ECL, *emitter-coupled logic*) es una familia de lógica digital no saturada. Dado que los transistores no se saturan, es posible lograr retardos de propagación de 1-2 ns. Esta familia de lógica tiene el retardo de propagación más bajo de todas y se le utiliza principalmente en sistemas que deben operar a gran velocidad. Su inmunidad al ruido y disipación de potencia, empero, son las peores de todas las familias de lógica disponibles.

En la figura 10-17 se observa un circuito básico típico de la familia ECL. Las salidas producen ambas funciones, OR y NOR. Cada entrada se conecta a la base de un transistor. Los dos niveles de voltaje aproximados son -0.8 V para el estado alto y -1.8 V para el estado bajo. El circuito consiste en un amplificador diferencial, una red de polarización compensada por temperatura y voltaje y una salida de emisor-seguidor. Las salidas de los emisores requieren un resistor de descenso (*pull-down*) para que fluya corriente. Esto se obtiene del resistor de entrada R_P de otra compuerta similar o de un resistor externo conectado a una fuente de voltaje negativo.

El circuito interno de polarización compensado por temperatura y voltaje proporciona un voltaje de referencia al amplificador diferencial. El voltaje de polarización V_{BB} se fija en -1.3 V, que es el punto medio de la oscilación lógica de la señal. Los diodos del divisor de voltaje, junto con $Q6$, crean un circuito que mantiene un valor constante de V_{BB} pese a cambios en la temperatura o el voltaje de alimentación. Cualquiera de las entradas de la fuente de poder se puede usar como tierra (GND). No obstante, el uso del nodo V_{CC} como tierra y $V_{EE} = -5.2$ V produce la inmunidad óptima al ruido.

Si cualquier entrada de la compuerta ECL es alta, se enciende el transistor correspondiente y $Q5$ se apaga. Una entrada de -0.8 V hace que el transistor conduzca y alimenta -1.6 V a los emisores de todos los transistores (la caída de V_{BE} en los transistores ECL es de 0.8 V). Puesto que $V_{BB} = -1.3$ V, el voltaje de base de $Q5$ es apenas 0.3 V más positivo que su emisor. $Q5$

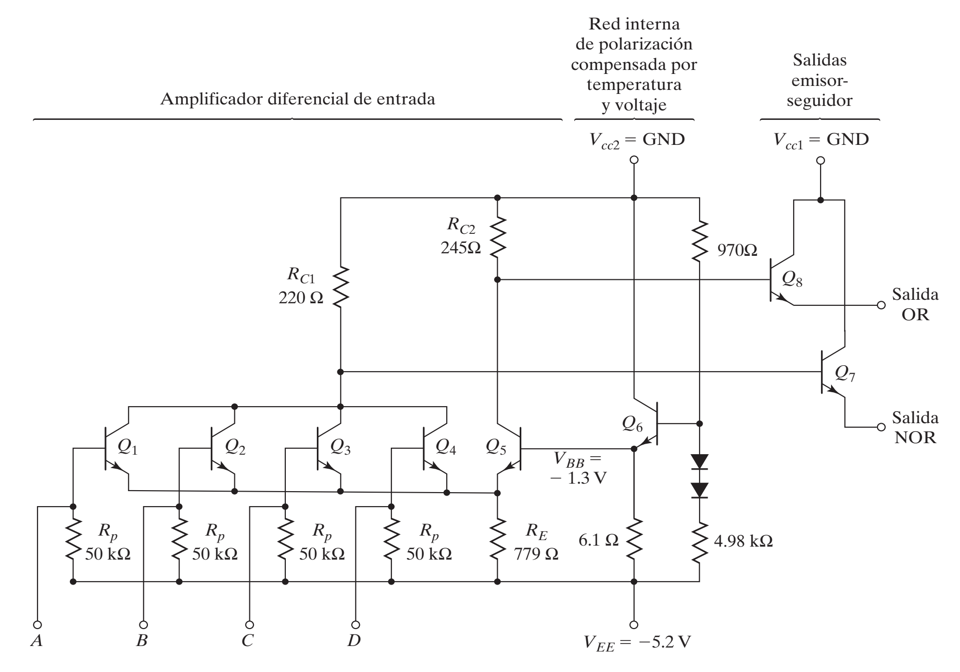


FIGURA 10-17
Compuerta ECL básica

está en corte porque su voltaje V_{BE} necesita por lo menos 0.6 V para comenzar a conducir. La corriente del resistor R_{C2} fluye a la base de $Q8$ (si hay un resistor de carga). Esta corriente es tan pequeña que la caída de voltaje en R_{C2} es insignificante. La salida OR de la compuerta está una caída de V_{BE} por debajo de tierra, o sea, -0.8 V, que es el estado alto. La corriente que fluye a través de R_{C1} y el transistor conductor produce una caída de aproximadamente 1 V por debajo de tierra (véase el problema 10-9). La salida NOR está una caída V_{BE} por debajo de este nivel, es decir, en -1.8 V, que es el estado bajo.

Si todas las entradas están en el nivel bajo, todos los transistores de entrada se apagan y $Q5$ conduce. El voltaje en el nodo de emisor común está una caída de V_{BE} por debajo de V_{BB}, o sea, en -2.1 V. Puesto que la base de cada entrada está en el nivel bajo de -1.8 V, cada unión emisor-base tiene únicamente 0.3 V y todos los transistores de entrada están en corte. R_{C2} extrae corriente a través de $Q5$, lo que causa una caída de voltaje de aproximadamente 1 V y hace que la salida OR esté una caída de V_{BE} por debajo de este nivel, en -1.8 V, que es el nivel bajo. La corriente en R_{C1} es insignificante y la salida NOR está una caída de V_{BE} por debajo de tierra, en -0.8 V, que es el nivel alto. Esto verifica las operaciones OR y NOR del circuito.

El retardo de propagación de la compuerta ECL es de 2 ns, y su disipación de potencia, 25 mW. Esto da un producto rapidez-potencia de 50, que es similar al del TTL Schottky. El margen de ruido es de aproximadamente 0.3 V, no tan bueno como el de la compuerta TTL. La compuerta TTL puede tener un abanico de salida grande gracias a la elevada impedancia de entrada del amplificador diferencial y la baja impedancia de salida del emisor-seguidor. La velocidad extremadamente alta de las señales hace que los alambres externos actúen como líneas de transmisión. Salvo en el caso de cables muy cortos de unos cuantos centímetros, las salidas ECL deben utilizar cables coaxiales con una terminación resistiva para reducir las reflexiones en las líneas.

El símbolo gráfico de la compuerta ECL se indica en la figura 10-18a). Tiene dos salidas: una para la función NOR y otra para la función OR. Es posible conectar las salidas de dos o más compuertas ECL para formar lógica alambrada. Como se observa en la figura 10-18b), una conexión alambrada *externa* de dos salidas NOR produce una función OR alambrada. En algunos CI ECL se utiliza una conexión alambrada *interna* de dos salidas OR para producir una lógica de AND alambrado (también llamada punto-AND). Esta propiedad podría aprovecharse cuando se usan compuertas ECL para formar las funciones OR-AND-INVERT y OR-AND.

10-7 METAL-ÓXIDO-SEMICONDUCTOR (MOS)

El transistor de efecto de campo (FET, *field-effect transistor*) es un transistor unipolar, ya que su funcionamiento depende del flujo de sólo un tipo de portador. Hay dos tipos de transistores de efecto de campo: el transistor de efecto de campo de unión (JFET, *junction FET*) y el me-

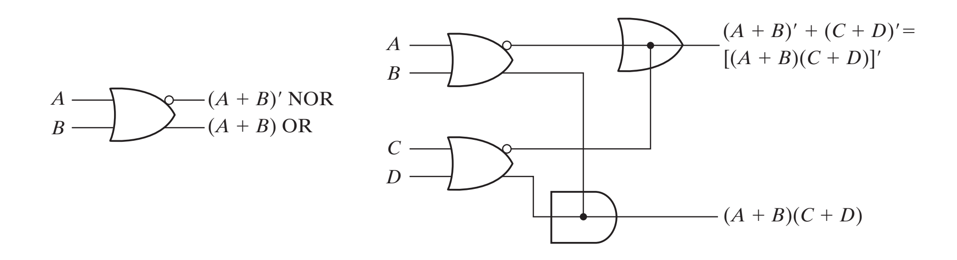
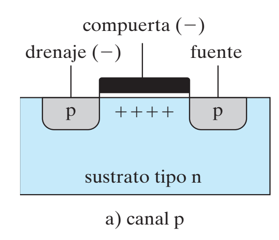
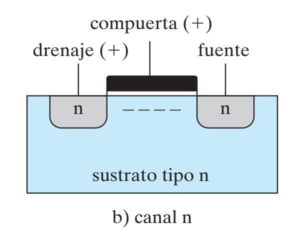


FIGURA 10-18
Símbolos gráficos de compuertas ECL



a) canal p



b) canal n

FIGURA 10-19
Estructura básica del transistor MOS

tal-óxido-semiconductor. El primero se usa en circuitos lineales, y el segundo, en circuitos digitales. Los transistores MOS se pueden fabricar en un área más reducida que los bipolares.

La estructura básica del transistor MOS se presenta en la figura 10-19. El MOS de canal *p* consiste en un sustrato levemente impurificado de silicio tipo *n*. Dos regiones se impurifican intensamente por difusión de impurezas tipo *p* para formar la *fuente* y el *drenaje*. La región entre las dos secciones tipo *p* actúa como *canal*. La *compuerta* es una placa de metal separada del canal por un dieléctrico aislado de dióxido de silicio. Un voltaje negativo (con respecto al sustrato) en la terminal de compuerta produce un campo eléctrico inducido en el canal que atrae del sustrato portadores tipo *p*. Al aumentar la magnitud del voltaje negativo en la compuerta, la región abajo de la compuerta acumula más portadores positivos, la conductividad aumenta y ya puede fluir corriente de la fuente al drenaje, a condición de que se mantenga una diferencia de voltaje entre estas dos terminales.

Hay cuatro tipos básicos de estructuras MOS. El canal puede ser tipo *p* o tipo *n*, dependiendo de si la mayoría de los portadores son huecos o electrones. El modo de operación puede ser de enriquecimiento o de agotamiento, dependiendo del estado de la región del canal que tiene voltaje de compuerta cero. Si en principio el canal está levemente impurificado con impurezas tipo *p* (canal difuso), existe un canal conductor con voltaje de compuerta cero y decimos que el dispositivo opera en modo de *agotamiento*. En este modo, fluye corriente a menos que el canal se agote aplicando un campo a la compuerta. Si la región abajo de la compuerta se deja inicialmente sin carga, el campo de compuerta deberá inducir un canal que permita fluir corriente. Así pues, el voltaje de compuerta enriquece la corriente de canal y decimos que el dispositivo opera en modo de *enriquecimiento*.

La fuente es la terminal por la que la mayoría de los portadores ingresa en la barra. El drenaje es la terminal por la que la mayoría de los portadores sale de la barra. En un MOS de canal *p*, la terminal de fuente está conectada al sustrato y se aplica un voltaje negativo a la terminal de drenaje. Cuando el voltaje de compuerta está por arriba de un voltaje de umbral V_T (unos −2 V), no fluye corriente en el canal y la trayectoria del drenaje a la fuente es como un circuito abierto. Cuando el voltaje de compuerta es lo bastante negativo por debajo de V_T, se forma un canal y los portadores tipo *p* fluyen de la fuente al drenaje. Los portadores tipo *p* son positivos y corresponden a un flujo de corriente positiva de la fuente al drenaje.

En el MOS de canal *n*, la terminal de fuente se conecta al sustrato y se aplica un voltaje positivo a la terminal de drenaje. Cuando el voltaje de compuerta está por debajo del voltaje de umbral V_T (unos 2 V), no fluye corriente en el canal. Cuando el voltaje de compuerta es lo bastante positivo por arriba de V_T como para formar un canal, portadores tipo *n* fluyen de la fuerte al drenaje. Los portadores tipo *n* son negativos, lo que corresponde a un flujo de corriente positiva del drenaje a la fuente. El voltaje de umbral podría variar entre 1 y 4 V, dependiendo del proceso específico empleado.

La figura 10-20 ilustra los símbolos gráficos para los transistores MOS. El símbolo para el transistor de enriquecimiento tiene una conexión de línea interrumpida entre la fuente y el drenaje. En este símbolo puede identificarse el sustrato y se le muestra conectado a la fuente. Un símbolo alterno omite el sustrato y coloca una flecha en la terminal de fuente para indicar la

drenaje
compuerta sustrato
fuente
D
G
S
a) canal p

drenaje
compuerta sustrato
fuente
D
G
S
b) canal n

FIGURA 10-20
Símbolos para transistores MOS

dirección del flujo de corriente *positiva* (de la fuente al drenaje en el canal *p* y del drenaje a la fuente en el canal *n*).

Debido a la construcción simétrica de la fuente y el drenaje, el transistor MOS puede operarse como dispositivo bilateral. Aunque normalmente se opera de modo que los portadores fluyan de la fuente al drenaje, hay circunstancias en las que conviene permitir el flujo de portadores del drenaje a la fuente (véase el problema 10-12).

Una ventaja del dispositivo MOS es que se le puede usar como transistor y también como resistor. Se obtiene un resistor del MOS polarizando de forma permanente la terminal de compuerta para que conduzca. Entonces, la razón del voltaje fuente-drenaje entre la corriente de canal determina el valor de la resistencia. Es factible construir resistores con diferentes valores fijando durante la fabricación la longitud y anchura del canal del dispositivo MOS.

En la figura 10-21 se aprecian tres circuitos lógicos que usan dispositivos MOS. En un MOS de canal *p*, el voltaje de alimentación V_{DD} es positivo (unos 5 V) para que pueda haber flujo de corriente positiva del drenaje a la fuente. Los dos niveles de voltaje son función del voltaje de umbral V_T. El nivel bajo puede ser cualquiera entre cero y V_T, y el alto varía entre V_T y V_{DD}. Las compuertas de canal *n* por lo regular emplean lógica positiva. Los circuitos de canal *p* utilizan un voltaje negativo en V_{DD} para que pueda haber flujo de corriente positiva de la fuente al drenaje. Los dos niveles de voltaje son negativos, por arriba y por debajo del voltaje de umbral negativo V_T. Las compuertas de canal *p* suelen utilizar lógica negativa.

El circuito inversor que aparece en la figura 10-21a) usa dos dispositivos MOS. *Q*1 actúa como resistor de carga, y *Q*2, como dispositivo activo. El resistor de carga MOS tiene su compuerta conectada a V_{DD}, lo que lo mantiene siempre en el estado de conducción. Cuando el voltaje de entrada es bajo (menor que V_T), Q2 se apaga. Puesto que *Q*1 siempre está encendido, el voltaje de salida es aproximadamente V_{DD}. Cuando el voltaje de entrada es alto (mayor que V_T), *Q*2 se enciende. Fluye corriente desde V_{DD} hasta *Q*2, pasando por el resistor de carga *Q*1. La geometría de los dos dispositivos MOS debe ser tal que la resistencia de *Q*2, al conducir, sea mucho menor que la resistencia de *Q*1, a fin de mantener la salida *Y* en un voltaje menor que V_T.

La compuerta NAND que se ilustra en la figura 10-21b) utiliza transistores en serie. Ambas entradas, *A* y *B*, deben ser altas para que todos los transistores conduzcan y la salida sea baja. Si cualquier entrada es baja, el transistor correspondiente se apagará y la salida será alta. Una vez más, la resistencia en serie formada por los dos dispositivos MOS activos debe ser mucho menor que la resistencia del resistor de carga MOS. La compuerta NOR que se muestra en la figura 10-21c) utiliza transistores en paralelo. Si cualquiera de las entradas es alta, el transistor correspondiente conduce y la salida es baja. Si todas las entradas son bajas, todos los transistores activos están apagados y la salida es alta.

10-8 MOS COMPLEMENTARIO (CMOS)

Los circuitos MOS complementario aprovechan el hecho de que es posible fabricar dispositivos tanto de canal *n* como de canal *p* en el mismo sustrato. Los circuitos CMOS constan de ambos tipos de dispositivos MOS interconectados para formar funciones lógicas. El circuito básico

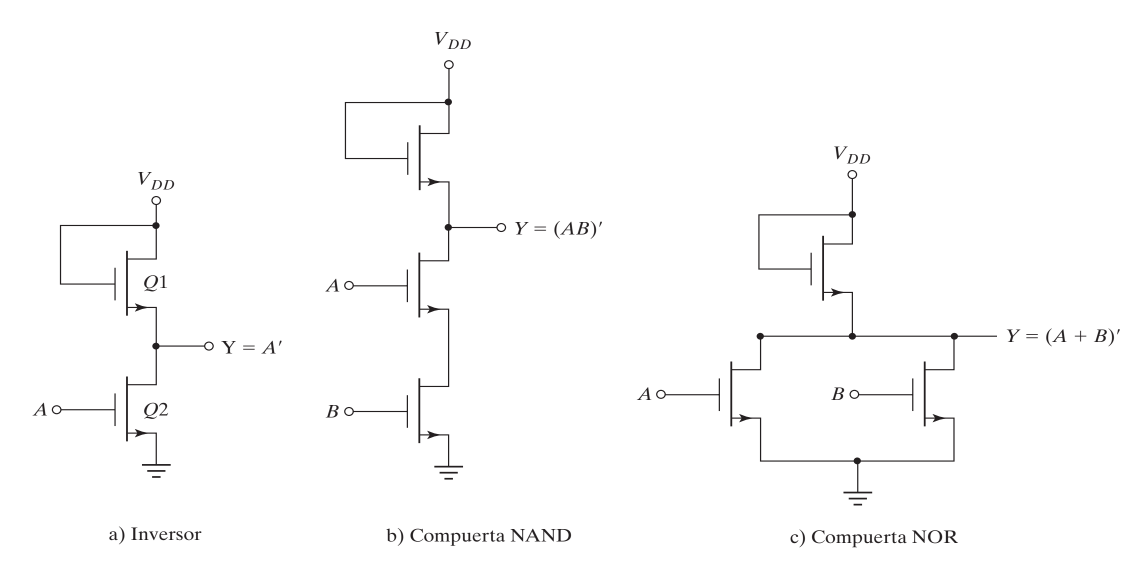


FIGURA 10-21
Circuitos de lógica MOS de canal *n*

es el inversor, que consiste en un transistor de canal *p* y un transistor de canal *n*, como se aprecia en la figura 10-22a). La terminal de fuente del dispositivo de canal *p* está en V_{DD}, y la terminal de fuente del dispositivo de canal *n* está en tierra. El valor de V_{DD} podría ser cualquiera entre +3 y +18 V. Los dos niveles de voltaje son 0 V para el nivel bajo y V_{DD} para el nivel alto (por lo regular 5 V).

Para entender el funcionamiento del inversor, es preciso repasar el comportamiento del transistor MOS, que se explicó en la sección anterior:

1. El MOS de canal *n* conduce cuando su voltaje de compuerta a fuente es positivo.
2. El MOS de canal *p* conduce cuando su voltaje de compuerta a fuente es negativo.
3. Ambos tipos se apagan si el voltaje de compuerta a fuente es cero.

Consideremos ahora el funcionamiento del inversor. Cuando la entrada es baja, ambas compuertas están en potencial cero. La entrada está en $-V_{DD}$ relativa a la fuente del dispositivo de canal *p*, y en 0 V relativa a la fuente del dispositivo de canal *n*. El resultado es que el dispositivo de canal *p* se enciende y el de canal *n* se apaga. En estas condiciones, hay una trayectoria de baja impedancia desde V_{DD} hasta la salida y una trayectoria de muy alta impedancia de la salida a tierra. Por tanto, el voltaje de salida se aproxima al nivel alto V_{DD} en condiciones de carga normal. Cuando la entrada es alta, ambas compuertas están en V_{DD} y la situación se invierte: el dispositivo de canal *p* está apagado y el de canal *n* encendido. El resultado es que la salida se acerca al nivel bajo de 0 V.

En la figura 10-22 se ilustran otras dos compuertas CMOS básicas. Una compuerta NAND de dos entradas consta de dos unidades tipo *p* en paralelo y dos unidades tipo *n* en serie, como se aprecia en la figura 10-22b). Si todas las entradas son altas, ambos transistores de canal *p* se apagan y ambos transistores de canal *n* se encienden. La salida tiene impedancia baja hacia tierra y produce un estado bajo. Si cualquier entrada es baja, el transistor de canal *n* asociado a ella se apaga y el transistor de canal *p* asociado se enciende. La salida se acopla a V_{DD} y pasa al estado alto. Es posible formar compuertas NAND de múltiples entradas colocando iguales cantidades de transistores tipo *p* y tipo *n* en paralelo y en serie, respectivamente, en una disposición similar a la de la figura 10-22b).

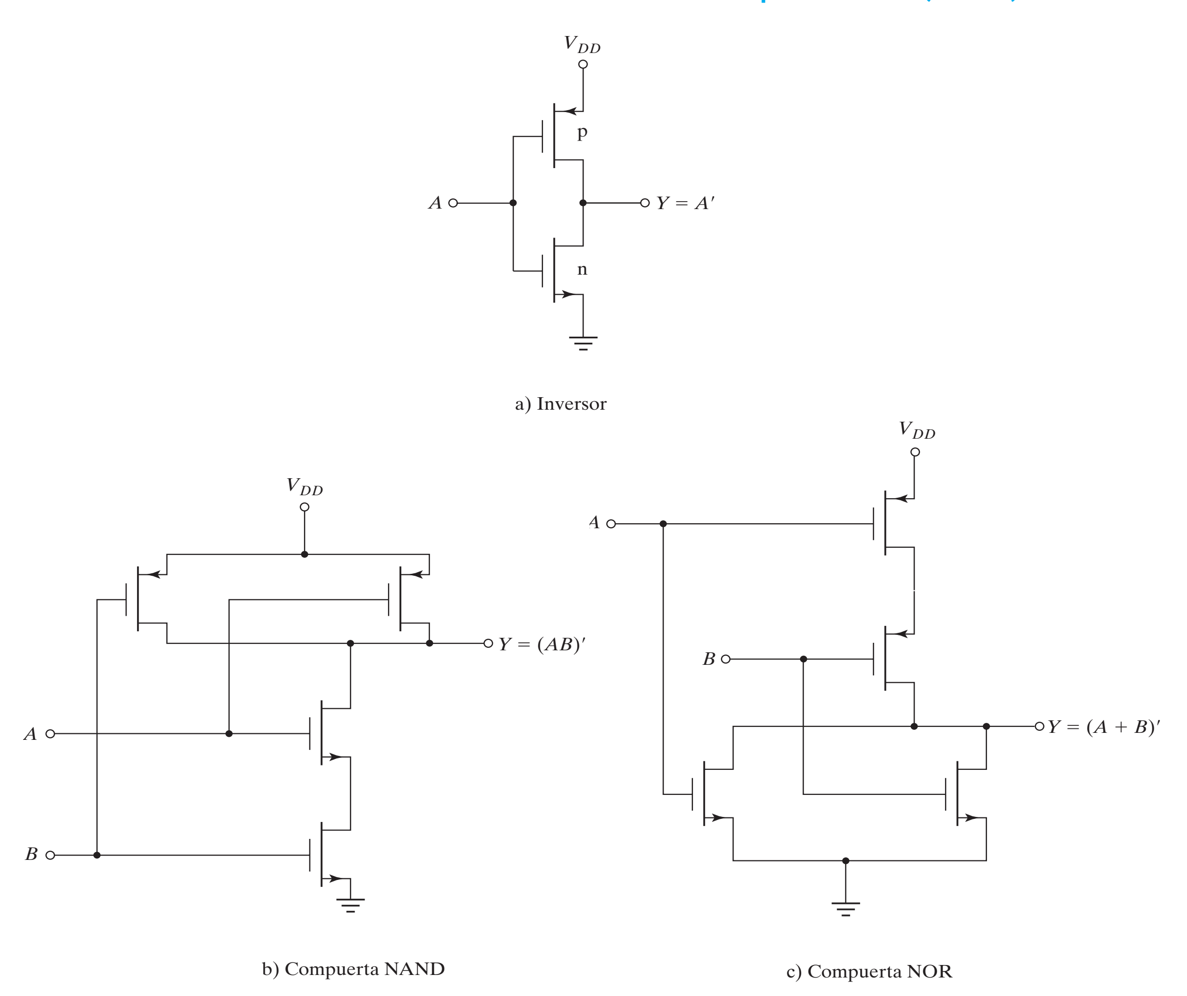


FIGURA 10-22
Circuitos de lógica CMOS

Una compuerta NOR de dos entradas consiste en dos unidades tipo *n* en paralelo y dos tipo *p* en serie, como se observa en la figura 10-22c). Cuando todas las entradas son bajas, ambas unidades de canal *p* están encendidas y las dos de canal *n* están apagadas. La salida se acopla a V_{DD} y pasa al estado alto. Si cualquier entrada está alta, el transistor de canal *p* asociado a ella se apaga y el de canal *n* se enciende. Esto conecta la salida a tierra, con lo que pasa al nivel bajo.

Los transistores MOS pueden verse como interruptores electrónicos que están conduciendo o están abiertos. Por ejemplo, el inversor CMOS puede verse como formado por dos interruptores, como se muestra en la figura 10-23a). La aplicación de un voltaje bajo a la entrada hace que el interruptor superior (*p*) se cierre y alimente un voltaje alto a la salida. La aplicación de un voltaje alto a la entrada hace que el interruptor inferior (*n*) se cierre y conecte la salida a tierra. Así, la salida V_{salida} es el complemento de la entrada $V_{entrada}$. En aplicaciones comerciales suelen utilizarse otros símbolos gráficos para los transistores MOS con objeto de subrayar el comportamiento lógico de los interruptores. Se han omitido las flechas que indican la dirección

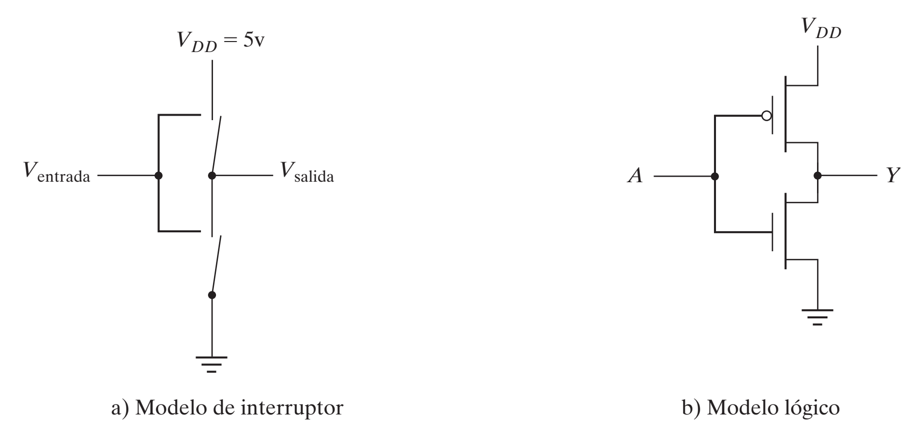


FIGURA 10-23
Inversor *CMOS*

del flujo de corriente. En vez de ello, la entrada de compuerta del transistor de canal *p* se dibuja con una burbuja de inversión en la terminal de compuerta para indicar que se habilita con un voltaje bajo. En la figura 10-23b) se ha vuelto a dibujar con estos símbolos el circuito inversor. Un 0 lógico en la entrada hace que el transistor superior conduzca y la salida sea 1 lógico. Un 1 lógico en la entrada habilita el transistor inferior, y la salida es 0 lógico.

Características del CMOS

Cuando un circuito lógico CMOS está en un estado estático, su disipación de potencia es muy baja. Ello se debe a que siempre hay un transistor apagado en la trayectoria cuando el estado del circuito no está cambiando. El resultado es que una compuerta CMOS típica tiene una disipación de potencia estática del orden de 0.01 mW. Cuando el circuito está cambiando de estado con una frecuencia de 1 MHz, la disipación de potencia aumenta a cerca de 1 mW, y a 10 MHz, a cerca de 5 mW.

La lógica CMOS por lo regular se especifica para una sola operación de fuente de poder dentro de un intervalo de voltaje de 3 a 18 V con un valor V_{DD} típico de 5 V. Si los CMOS se operan con un voltaje de alimentación mayor, se reduce el retardo de propagación y mejora el margen de ruido, pero la disipación de potencia aumenta. El retardo de propagación con $V_{DD} = 5$ V varía entre 5 y 20 ns, dependiendo del tipo utilizado. El margen de ruido suele ser de aproximadamente un 40% del voltaje de alimentación. El abanico de salida de las compuertas CMOS es de alrededor de 30 cuando se operan con una frecuencia de 1 MHz. El abanico de salida disminuye al aumentar la frecuencia de operación.

Hay varias series en la familia de lógica digital CMOS. La serie 74C es compatible en cuanto a terminales y función con los dispositivos TTL que llevan el mismo número. Por ejemplo, el CI CMOS tipo 74C04 tiene seis inversores con la misma configuración de terminales que el TTL tipo 7404. La serie CMOS 74HC de alta velocidad es una mejora de la serie 74C que tiene una velocidad de conmutación 10 veces mayor. La serie 74HCT es eléctricamente compatible con los CI TTL. Esto implica que el circuito de esta serie se puede conectar a entradas y salidas de CI TTL sin necesidad de circuitos de interfaz adicionales. Las versiones más nuevas de CMOS son la serie 74VHC y su versión compatible con TTL, 74VHCT.

El proceso de fabricación CMOS es más sencillo que el de TTL y permite empacar componentes con mayor densidad. Esto permite colocar más circuitos en un área dada de silicio, lo que reduce el costo por función. Esta propiedad, junto con la baja disipación de potencia, buena inmunidad al ruido y retardo de propagación razonable, hace que CMOS sea el estándar más popular como familia de lógica digital.

10-9 CIRCUITOS DE COMPUERTA DE TRANSMISIÓN CMOS

Un circuito CMOS especial con que no cuentan las demás familias de lógica digital es la *compuerta de transmisión* (*transmission gate*). En esencia, una compuerta de transmisión es un interruptor electrónico controlado por un nivel lógico de entrada. Simplifica la construcción de diversos componentes digitales que se fabrican con tecnología CMOS.

La figura 10-24a) muestra el circuito básico de la compuerta de transmisión. Consiste en un transistor MOS de canal *n* y uno de canal *p* conectados en paralelo.

El sustrato de canal *n* se conecta a tierra, y el de canal *p*, a V_{DD}. Cuando la compuerta *N* está en V_{DD} y la compuerta *P* está en tierra, ambos transistores conducen y hay una trayectoria cerrada entre la entrada *X* y la salida *Y*. Cuando la compuerta *N* está en tierra y la *P* está en V_{DD}, ambos transistores están apagados y hay un circuito abierto entre *X* y *Y*. La figura 10-24b) presenta el diagrama de bloques de la compuerta de transmisión. Observe que la terminal de la com-

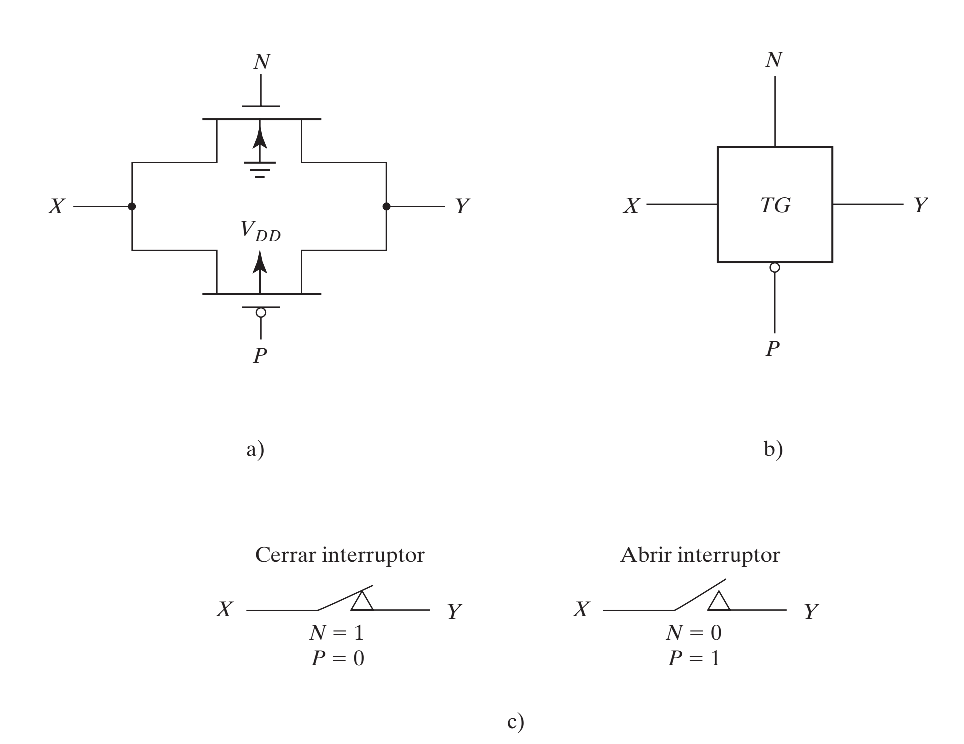


FIGURA 10-24
Compuerta de transmisión (TG)

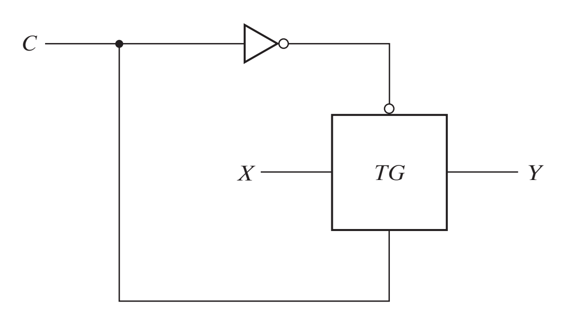


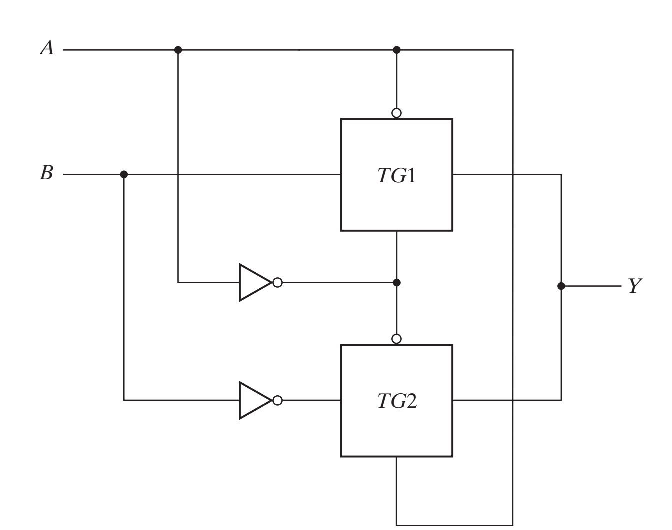
FIGURA 10-25
Interruptor bilateral

puerta de canal *p* se marca con el símbolo de negación. La figura 10-24c) ilustra el comportamiento del interruptor en términos de asignación de lógica positiva, donde V_{DD} equivale a 1 lógico y tierra equivale a 0 lógico.

La compuerta de transmisión suele conectarse a un inversor, como se indica en la figura 10-25. Este tipo de acomodo se conoce como *interruptor bilateral*. La entrada de control *C* se conecta directamente a la compuerta de canal *n*, y su inverso, a la compuerta de canal *p*. Cuando $C = 1$, el interruptor está cerrado y crea una trayectoria entre *X* y *Y*.

Es posible construir diversos circuitos empleando la compuerta de transmisión. Para ilustrar su utilidad como componente en la familia CMOS, presentaremos tres ejemplos de circuitos.

La compuerta OR exclusivo se puede construir con dos compuertas de transmisión y dos inversores, como se muestra en la figura 10-26. La entrada *A* controla las trayectorias de las compuertas de transmisión y la entrada *B* está conectada a la salida *Y* a través de las compuertas. Cuando la entrada *A* es 0, la compuerta de transmisión *TG*1 está cerrada y la salida *Y* es igual



A	*B*	*TG*1	*TG*2	*Y*
0	0	cerrar	abrir	0
0	1	cerrar	abrir	1
1	0	abrir	cerrar	1
1	1	abrir	cerrar	0

FIGURA 10-26
OR exclusivo construido con compuertas de transmisión

a la entrada B. Cuando la entrada A es 1, $TG2$ está cerrada y la salida Y es igual al complemento de la entrada B. El resultado es la tabla de verdad del OR exclusivo, como indica la tabla de la figura 10-26.

Otro circuito que puede construirse con compuertas de transmisión es el multiplexor. En la figura 10-27 se aprecia un multiplexor de 4 líneas a 1 implementado con compuertas de transmisión. El circuito TG crea una trayectoria de transmisión entre sus líneas horizontales de entrada y salida cuando las dos entradas verticales de control tienen el valor 1 en la terminal sin burbuja y 0 en la terminal con burbuja. Con una polaridad opuesta en las entradas de control, la trayectoria se desconecta y el circuito se comporta como interruptor abierto. Las dos entradas de selección, S_1 y S_0, controlan la trayectoria de transmisión en los circuitos TG. Dentro de cada

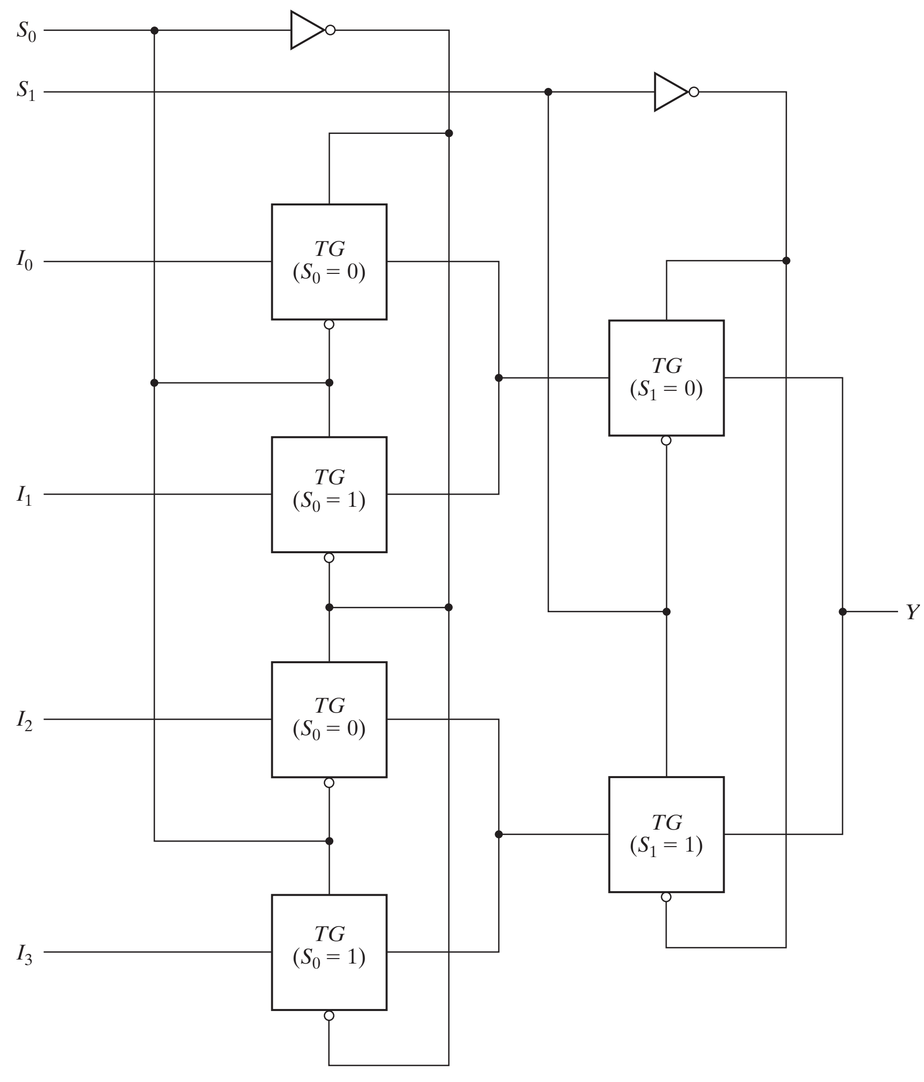


FIGURA 10-27
Multiplexor con compuertas de transmisión

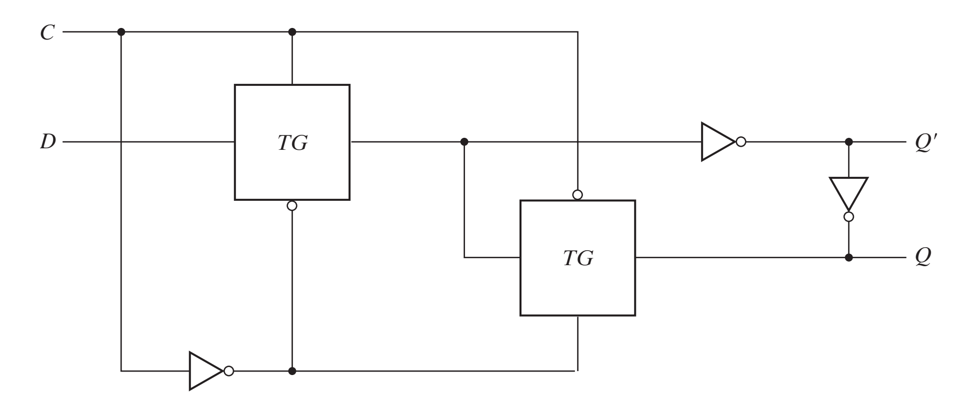


FIGURA 10-28
Latch *D* con compuerta implementado con compuertas de transmisión

rectángulo se ha anotado la condición para que el interruptor de compuerta de transmisión esté cerrado. Así pues, si $S_0 = 0$ y $S_1 = 0$, hay una trayectoria cerrada de la entrada I_0 a la salida *Y* a través de las dos *TG* marcadas con $S_0 = 0$ y $S_1 = 0$. Las otras tres entradas quedan desconectadas de la salida por uno de los otros circuitos *TG*.

El flip-flop *D* sensible al nivel, comúnmente conocido como latch *D* con compuerta, se puede construir con compuertas de transmisión, como se aprecia en la figura 10-28. La entrada *C* controla dos compuertas de transmisión *TG*. Cuando $C = 1$, la *TG* conectada a la entrada *D* tiene una trayectoria cerrada, mientras que la conectada a la salida *Q* tiene una trayectoria abierta. Esto produce un circuito equivalente desde la entrada *D* hasta la salida *Q*, pasando por dos inversores. Por tanto, la salida sigue a la entrada de datos en tanto *C* se mantenga activa. Cuando *C* cambia a 0, la primera *TG* desconecta la entrada *D* del circuito y la segunda *TG* crea una trayectoria cerrada entre los dos inversores en la salida. Así, el valor que estaba presente en la entrada *D* cuando *C* cambió de 1 a 0 se conserva en la salida *Q*.

Podemos construir un flip-flop *D* amo-esclavo con dos circuitos como el de la figura 10-28. El primero es el amo, y el segundo, el esclavo. Por tanto, es posible construir un flip-flop *D* amo-esclavo con cuatro compuertas de transmisión y seis inversores.

10-10 MODELADO EN EL NIVEL DE INTERRUPTORES CON HDL

CMOS es la familia de lógica digital dominante empleada en circuitos integrados. Por definición, CMOS es una conexión complementaria de un transistor NMOS y uno PMOS. Los transistores MOS pueden verse como interruptores electrónicos que conducen, o bien, están abiertos. Al especificar las conexiones entre interruptores MOS, el diseñador puede describir un circuito digital construido con CMOS. En Verilog HDL, este tipo de descripción se llama modelado en el nivel de interruptores.

Los dos tipos de interruptores MOS se especifican en Verilog HDL con las palabras clave **nmos** y **pmos.** Creamos ejemplares de ellos especificando las tres terminales del transistor, que se observan en la figura 10-20.

```
nmos (drenaje, fuente, compuerta);
pmos (drenaje, fuente, compuerta);
```

Los interruptores se consideran primitivos, así que el uso de un nombre de ejemplar es opcional.

Es preciso especificar las conexiones a una fuente de poder (V_{DD}) y a tierra cuando se diseñan los circuitos MOS. La alimentación y tierra se definen con las palabras clave **supply1** y **supply0,** y se especifican con estos enunciados:

```
supply1 PWR;
supply0 GRD;
```

Las fuentes tipo **supply1** equivalen a V_{DD} y tienen un valor de 1 lógico. Las fuentes tipo **supply0** equivalen a una conexión a tierra y tienen un valor de 0 lógico.

La descripción del inversor CMOS de la figura 10-22a) se realiza en el ejemplo HDL 10-1. Primero se declaran la entrada, la salida y las dos fuentes de alimentación. El módulo crea un ejemplar de transistor PMOS y uno de NMOS. La salida *Y* es común a ambos transistores en sus terminales de drenaje. La entrada también es común a ambos transistores en sus terminales de compuerta. La terminal de fuente del transistor PMOS se conecta a PWR, y la del transistor NMOS, a GRD.

El segundo módulo del ejemplo 10-2 describe el circuito NAND CMOS de dos entradas de la figura 10-22b). Hay dos transistores PMOS conectados en paralelo, con sus terminales de fuente conectadas a PWR. Hay dos transistores NMOS conectados en serie con una terminal común *W*1. El drenaje del primer NMOS se conecta a la salida, y la fuente del segundo NMOS se conecta a tierra.

Ejemplo HDL 10-1

```
----------------------------------------------------
//Inversor CMOS de la figura 10-22a)
module inverter (Y,A);
   input A;
   output Y;
   supply1  PWR;
   supply0  GRD;
   pmos (Y,PWR,A);   //(Drenaje, fuente, compuerta)
   nmos (Y,GRD,A);   //(Drenaje, fuente, compuerta)
endmodule
----------------------------------------------------
```

Ejemplo HDL 10-2

```
----------------------------------------------------
//NAND CMOS de 2 entradas figura 10-22b)
module NAND2 (Y,A,B);
   input A,B;
   output Y;
   supply1  PWR;
   supply0  GRD;
   wire W1;              //Terminal entre dos nmos
   pmos (Y,PWR,A);       //Fuente conectada a Vdd
   pmos (Y,PWR,B);       //Conexión en paralelo
   nmos (Y,W1,A);        //Conexión en serie
   nmos (W1,GRD,B);      //Fuente conectada a tierra
endmodule
----------------------------------------------------
```

Compuerta de transmisión

En Verilog HDL se crea un ejemplar de compuerta de transmisión con la palabra clave **cmos.** Tiene una salida, una entrada y dos señales de control, como se indica en la figura 10-24. Se le denomina interruptor **cmos.** El código pertinente es:

```
cmos(salida, entrada, controln, controlp); //Descripción general
cmos(Y,X,N,P);  //Compuerta de transmisión de la figura 10-24b)
```

El controln y el controlp normalmente son uno el complemento del otro. El interruptor **cmos** no necesita fuentes de poder porque V_{DD} y tierra están conectados a los sustratos de los transistores MOS. Las compuertas de transmisión son útiles para construir multiplexores y flip-flops con circuitos CMOS.

El ejemplo HDL 10-3 ilustra la descripción de un circuito con interruptores **cmos.** El circuito OR exclusivo de la figura 10-26 tiene dos compuertas de transmisión y dos inversores. Se crean ejemplares de los dos inversores con el módulo de un inversor CMOS. Se crean ejemplares de los dos interruptores **cmos** sin nombre, porque se les considera primitivos. Se incluye un módulo de prueba para probar el funcionamiento del circuito. Al aplicar todas las combinaciones posibles de las dos entradas, el simulador verifica el funcionamiento del circuito OR exclusivo. El resultado de la simulación es:

$A = 0 \quad B = 0 \quad Y = 0$

$A = 0 \quad B = 1 \quad Y = 1$

$A = 1 \quad B = 0 \quad Y = 1$

$A = 1 \quad B = 1 \quad Y = 0$

Ejemplo HDL 10-3

```
//XOR con interruptores CMOS figura 10-25
module SXOR (A,B,Y);
   input A,B;
   output Y;
   wire Anot, Bnot;
//Crear ejemplar de inversor
   inverter v1 (Anot,A);
   inverter v2 (Bnot,B);
//Crear ejemplar de interruptor cmos
   cmos (Y,B,Anot,A);   //(salida, entrada, controln, controlp)
   cmos (Y,Bnot,A,Anot);
endmodule


//Inversor CMOS figura 10-22a)
module inverter (Y,A);
   input A;
   output Y;
   supply1  PWR;
   supply0  GRD;
   pmos (Y,PWR,A);      //(Drenaje, fuente, compuerta)
   nmos (Y,GRD,A);      //(Drenaje, fuente, compuerta)
endmodule


//Estímulo para probar SXOR
module test_SXOR;
   reg A,B;
   wire Y;
//Crear ejemplar de SXOR
   SXOR X1 (A,B,Y);
//Aplicar tabla de verdad
   initial
     begin
          A=1'b0; B=1'b0;
       #5 A=1'b0; B=1'b1;
       #5 A=1'b1; B=1'b0;
       #5 A=1'b1; B=1'b1;
     end
//Exhibir resultados
   initial
    $monitor ("A =%b  B= %b  Y =%b",A,B,Y);
endmodule
```

PROBLEMAS

10-1 Las siguientes son las especificaciones de las compuertas NAND cuádruples de dos entradas Schottky TTL 74S00. Calcule el abanico de salida, la disipación de potencia, el retardo de propagación y el margen de ruido de la compuerta NAND Schottky.

Parámetro	Nombre	Valor
V_{CC}	Voltaje de alimentación	5 V
I_{CCH}	Corriente de alimentación de alto nivel (cuatro compuertas)	10 mA
I_{CCL}	Corriente de alimentación de bajo nivel (cuatro compuertas)	20 mA
V_{OH}	Voltaje de salida de alto nivel (mín)	2.7 V
V_{OL}	Voltaje de salida de bajo nivel (máx)	0.5 V
V_{IH}	Voltaje de entrada de alto nivel (mín)	2 V
V_{IL}	Voltaje de entrada de bajo nivel (máx)	0.8 V
I_{OH}	Corriente de salida de alto nivel (máx)	1 mA
I_{OL}	Corriente de salida de bajo nivel (máx)	20 mA
I_{IH}	Corriente de entrada de alto nivel (máx)	0.05 mA
I_{IL}	Corriente de entrada de bajo nivel (máx)	2 mA
t_{PLH}	Retardo de bajo a alto	3 ns
t_{PHL}	Retardo de alto a bajo	3 ns

10-2 a) Determine el voltaje de salida de alto nivel de la compuerta RTL para un abanico de salida de 5. b) Determine el voltaje mínimo de entrada necesario para llevar a saturación un transistor RTL cuando $h_{FE} = 20$. c) Con base en los resultados de a) y b), determine el margen de ruido de la compuerta RTL cuando la entrada es alta y el abanico de salida es 5.

10-3 Demuestre que el transistor de salida de la compuerta DTL de la figura 10-9 entra en saturación cuando todas las entradas son altas. Suponga $h_{FE} = 20$.

10-4 Conecte la salida *Y* de la compuerta DTL que se muestra en la figura 10-9 a *N* entradas de otras compuertas similares. Suponga que el transistor de salida está saturado y que su corriente de base es de 0.44 mA. Sea $h_{FE} = 20$.

a) Calcule la corriente en el resistor de 2kΩ.

b) Calcule la corriente que proviene de cada entrada conectada a la compuerta.

c) Calcule la corriente de colector total en el transistor de salida, en función de *N*.

d) Encuentre el valor de *N* que mantiene al transistor en saturación.

e) ¿Qué abanico de salida tiene la compuerta?

10-5 Suponga que todas las entradas de la compuerta TTL de colector abierto de la figura 10-11 están en el estado alto de 3 V.

a) Determine los voltajes en la base, el colector y el emisor de todos los transistores.

b) Determine el h_{FE} mínimo de $Q2$ que garantiza que este transistor se saturará.

c) Calcule la corriente de base de $Q3$.

d) Suponga que el h_{FE} mínimo de $Q3$ es 6.18. ¿Qué corriente máxima puede tolerarse en el colector para garantizar la saturación de $Q3$?

e) ¿Qué valor mínimo de R_L puede tolerarse para garantizar la saturación de $Q3$?

10-6 a) Utilizando los transistores de salida reales de dos compuertas TTL de colector abierto, demuestre (con una tabla de verdad) que, cuando se conectan juntas a un resistor externo y a V_{CC}, la conexión alambrada produce una función AND.

b) Demuestre que dos inversores TTL de colector abierto conectados entre sí producen la función NOR.

10-7 En la sección 10-5 se dijo que las salidas de totem pole no deben conectarse entre sí para formar lógica alambrada. Para ver por qué esto es prohibitivo, conecte dos de esos circuitos y suponga que la salida de una compuerta está en el estado alto, y la de la otra, en el estado bajo. Demuestre que la corriente de carga (que es la suma de las corrientes de base y de colector del transistor saturado $Q4$ de la figura 10-14) es de aproximadamente 32 mA. Compare este valor con la corriente de carga recomendada en el estado alto, que es de 0.4 mA.

10-8 Para las condiciones siguientes, indique qué transistores están apagados y cuáles están conduciendo, en la compuerta TTL de tres estados de la figura 10-16c). (Para $Q1$ y $Q6$, sería necesario numerar por separado los estados en las uniones base-emisor y base-colector.)

a) Cuando C es baja y A es baja.
b) Cuando C es baja y A es alta.
c) Cuando C es alta.

¿Qué estado tiene la salida en cada caso?

10-9 Calcule la corriente de emisor I_E en R_E en la compuerta ECL de la figura 10-17 cuando

a) Por lo menos una entrada es alta (−0.8 V).
b) Todas las entradas son bajas (−1.8 V).
c) Suponga ahora que $I_C = I_E$. Calcule la caída de voltaje en el resistor de colector en cada caso y demuestre que es aproximadamente 1 V, como se requiere.

10-10 Calcule el margen de ruido de la compuerta ECL.

10-11 Utilizando las salidas NOR de dos compuertas ECL, demuestre que, cuando se conectan juntas a un resistor externo y una fuente de voltaje negativo, la condición alambrada produce una función OR.

10-12 El transistor MOS es bilateral, es decir, la corriente puede fluir de la fuente al drenaje o del drenaje a la fuente. Aprovechando esta propiedad, deduzca un circuito que implemente la función booleana

$$Y = (AB + CD + AED + CEB)'$$

utilizando seis transistores MOS.

10-13 a) Muestre el circuito de una compuerta NAND de cuatro entradas empleando transistores CMOS. b) Repita con una compuerta NOR de cuatro entradas.

10-14 Construya un circuito NOR exclusivo con dos inversores y dos compuertas de transmisión.

10-15 Construya un multiplexor de 8 líneas a 1 empleando compuertas de transmisión e inversores.

10-16 Dibuje el diagrama lógico de un flip-flop D amo-esclavo empleando compuertas de transmisión e inversores.

10-17 Escriba un conjunto de pruebas para el circuito NAND del ejemplo HDL 10-2. La simulación deberá verificar la tabla de verdad de la compuerta.

REFERENCIAS

1. TOCCI, R. J. y N. S. WIDMER. 2001. *Digital Systems Principles and Applications*, 8a. ed. Upper Saddle River, NJ: Prentice-Hall.

2. WESTE, N. E. y K. ESHRAGHIAN. 1993. *Principles of CMOS VLSI design: A System Perspective*, 2a. ed. Reading, MA: Addison-Wesley.

3. WAKERLY, J. F. 2000. *Digital Design: Principles and Practices*, 3a. ed. Upper Saddle River, NJ: Prentice-Hall.

4. HODGES, D. A. y H. G. JACKSON. 1988. *Analysis and Design of Digital Integrated Circuits*, 2a. ed. Nueva York: McGraw-Hill.

5. 1988. *The TTL Logic Data Book*. Dallas: Texas Instruments.

6. 1994. *CMOS Logic Data Book*. Dallas: Texas Instruments.

7. CILETTI, M. D. 1999. *Modeling, Synthesis and Rapid Prototyping with Verilog HDL*. Upper Saddle River, NJ: Prentice-Hall.

11 Experimentos de laboratorio

11-0 INTRODUCCIÓN A EXPERIMENTOS

En este capítulo se incluyen 18 experimentos de laboratorio sobre circuitos digitales y diseño lógico, con el fin de proporcionar algo de experiencia práctica al estudiante que usa este libro. Los circuitos digitales pueden construirse utilizando circuitos integrados (CI) estándar montados en tablas de pruebas que se arman fácilmente en el laboratorio. Los experimentos se han ordenado según el material presentado en el libro. La última sección consta de varios suplementos con sugerencias acerca del uso de Verilog HDL para simular y probar los circuitos digitales presentados en los experimentos.

Una tabla de pruebas lógicas apropiada para efectuar los experimentos deberá contar con el equipo siguiente:

1. Lámparas indicadoras LED (diodo emisor de luz).
2. Interruptores de dos posiciones para suministrar señales de 1 y 0 lógicos.
3. Pulsadores con botones accionadores y circuitos antirrebote para generar pulsos individuales.
4. Un generador de pulsos de reloj con por lo menos dos frecuencias: una baja de aproximadamente un pulso por segundo para observar cambios lentos en señales digitales, y una frecuencia más alta para observar formas de onda en un osciloscopio.
5. Una fuente de poder de 5 V.
6. Bases para montar los CI.
7. Alambre sólido para conectar y pinzas de corte para pelar y cortar los alambres.

Varios fabricantes ofrecen equipos de capacitación en lógica digital que incluyen los materiales requeridos. Tales equipos contienen lámparas LED, interruptores, pulsadores, un reloj variable, fuente de poder y bases para montar CI. Podrían necesitarse tablas de prueba extendidas con más bases libres de soldar e interruptores y lámparas insertables.

Otros equipos necesarios son un osciloscopio de dos canales (para los experimentos 1, 2, 8 y 15), una punta de prueba lógica para depuración y varios CI. Los CI que se requieren para los experimentos pertenecen a la serie 7400 TTL o CMOS.

Los circuitos integrados que se usarán en los experimentos se clasifican como circuitos de integración a pequeña escala (SSI) o a mediana escala (MSI). Los circuitos SSI contienen compuertas o flip-flops individuales, mientras que los MSI realizan funciones digitales específicas. Los ocho CI de compuertas SSI que se requieren para los experimentos se muestran en la figura 11-1. Incluyen compuertas NAND, NOR, AND, OR y XOR de dos entradas, inversores y compuertas NAND de tres y cuatro entradas. La asignación de las terminales de las compuertas se indica en el diagrama. Las terminales están numeradas del 1 al 14. La terminal 14 está marcada V_{CC}, y la 7, *GND* (tierra). Éstas son las terminales de alimentación que deben conectarse a una fuente de poder de 5 V para funcionar correctamente. Cada CI se reconoce por su número de identificación; por ejemplo, las compuertas NAND de dos entradas están en el CI cuyo número es 7400.

El lector encontrará descripciones detalladas de los circuitos MSI en los libros de datos publicados por los fabricantes. La mejor forma de adquirir experiencia con los circuitos MSI comerciales es estudiando su descripción en un libro de datos que proporciona información completa acerca de las características internas, externas y eléctricas de los circuitos integrados. Diversas compañías de semiconductores publican libros de datos para la serie 7400. Los circuitos MSI que se requieren para los experimentos se presentarán y explicarán la primera vez que se usen. El funcionamiento del circuito se explicará en términos de circuitos similares estudiados en capítulos anteriores. La información dada en este capítulo acerca de los circuitos MSI deberá bastar para realizar correctamente los experimentos. No obstante, siempre es preferible consultar un libro de datos, pues proporciona una descripción más detallada de los circuitos.

A continuación ilustraremos el método de presentación de circuitos MSI que hemos adoptado aquí. Haremos esto con un ejemplo específico que presenta el CI de contador de rizo, tipo 7493. Este CI se usa en el experimento 1 y en experimentos subsiguientes para generar una sucesión de números binarios con los cuales verificar el funcionamiento de circuitos combinacionales.

La información referente al CI 7493, contenida en el libro de datos, se presenta en las figuras 11-2a) y b). La parte a) corresponde a un diagrama del circuito lógico interno y su conexión hacia terminales externas. Se asignan símbolos de letra y números de terminal a todas las entradas y salidas. En la parte b) se observa la organización física del CI con la asignación de nombres de señal a las 14 terminales. Las terminales que el circuito no utiliza llevan la indicación *NC* (no hay conexión). El CI se inserta en una base, y se conectan cables a las diversas terminales a través de las terminales de la base. En los esquemas de este capítulo se presenta el CI en forma de diagrama de bloques, como en la figura 11-2c). El número de CI (7493 en este caso) se escribirá dentro del bloque. Todas las terminales de entrada se colocarán a la izquierda del bloque, y las de salida, a la derecha. Los símbolos de las señales, como *A*, *R*1 y *QA*, se escribirán dentro del bloque, mientras que los números de terminal externa, como 14, 2 y 12, se escribirán a lo largo de las líneas externas. V_{CC} y *GND* son las terminales de alimentación conectadas a las agujas 5 y 10. El tamaño del bloque podría variar para dar cabida a todas las terminales de entrada y salida. Ocasionalmente, podrían colocarse entradas o salidas en el lado superior o inferior del bloque, por conveniencia.

El funcionamiento del circuito es similar al contador de rizo de la figura 6-8a), con un despeje asincrónico de cada flip-flop. Cuando la entrada *R*1 o la *R*2, o ambas, son 0 lógico (tierra), todos los despejes asincrónicos son 1 y quedan inhabilitados. Para poner en ceros los cuatro flip-flops, la salida de la compuerta NAND debe ser 0. Esto se logra alimentando 1 lógico (aproximadamente 5 V) a ambas entradas, *R*1 y *R*2. Observe que las entradas *J* y *K* carecen de conexiones. Es característico de los circuitos TTL que una terminal de entrada sin conexión externa produzca una señal equivalente a 1 lógico. Observe también que la salida *QA* no está conectada internamente a la entrada *B*.

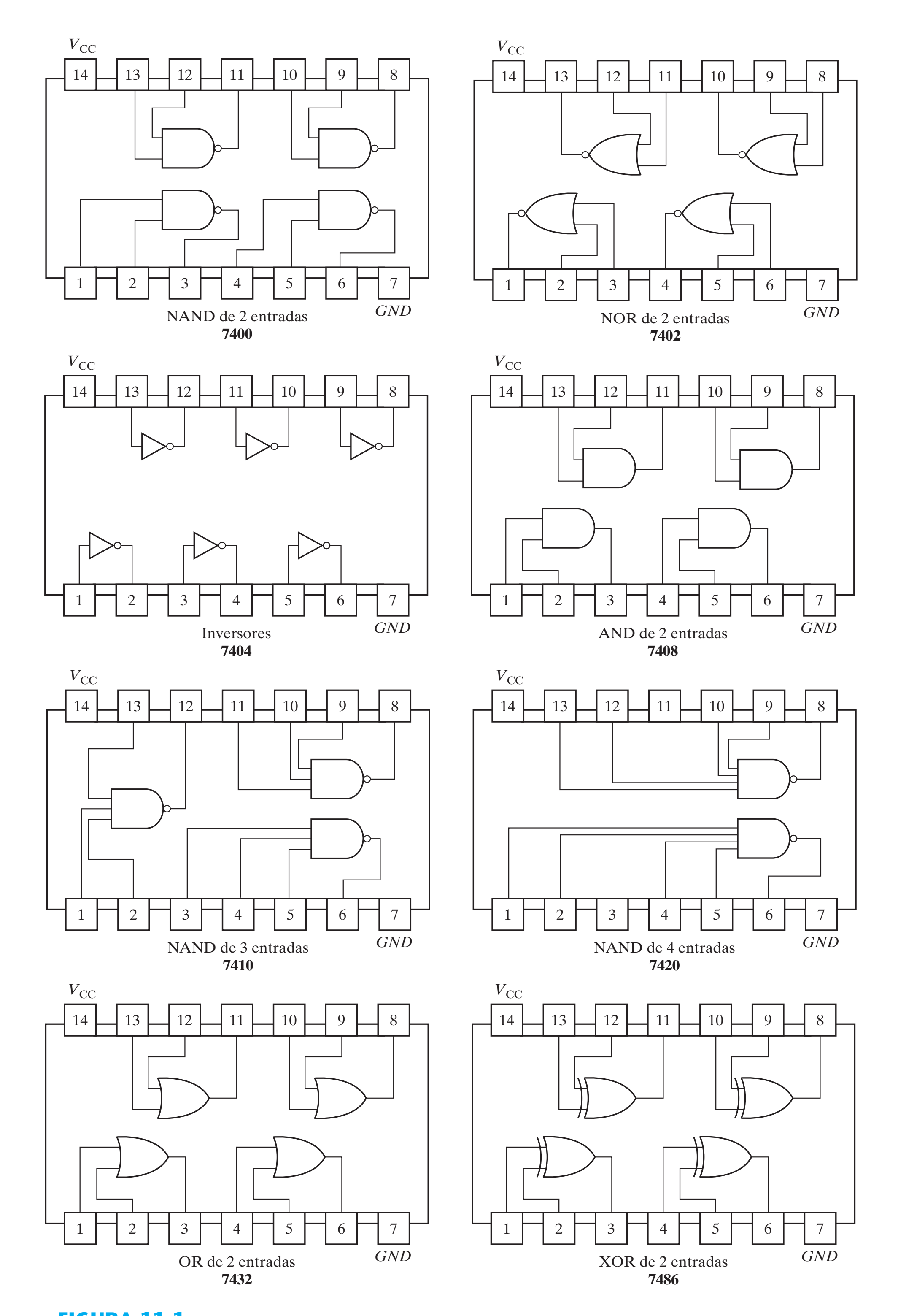
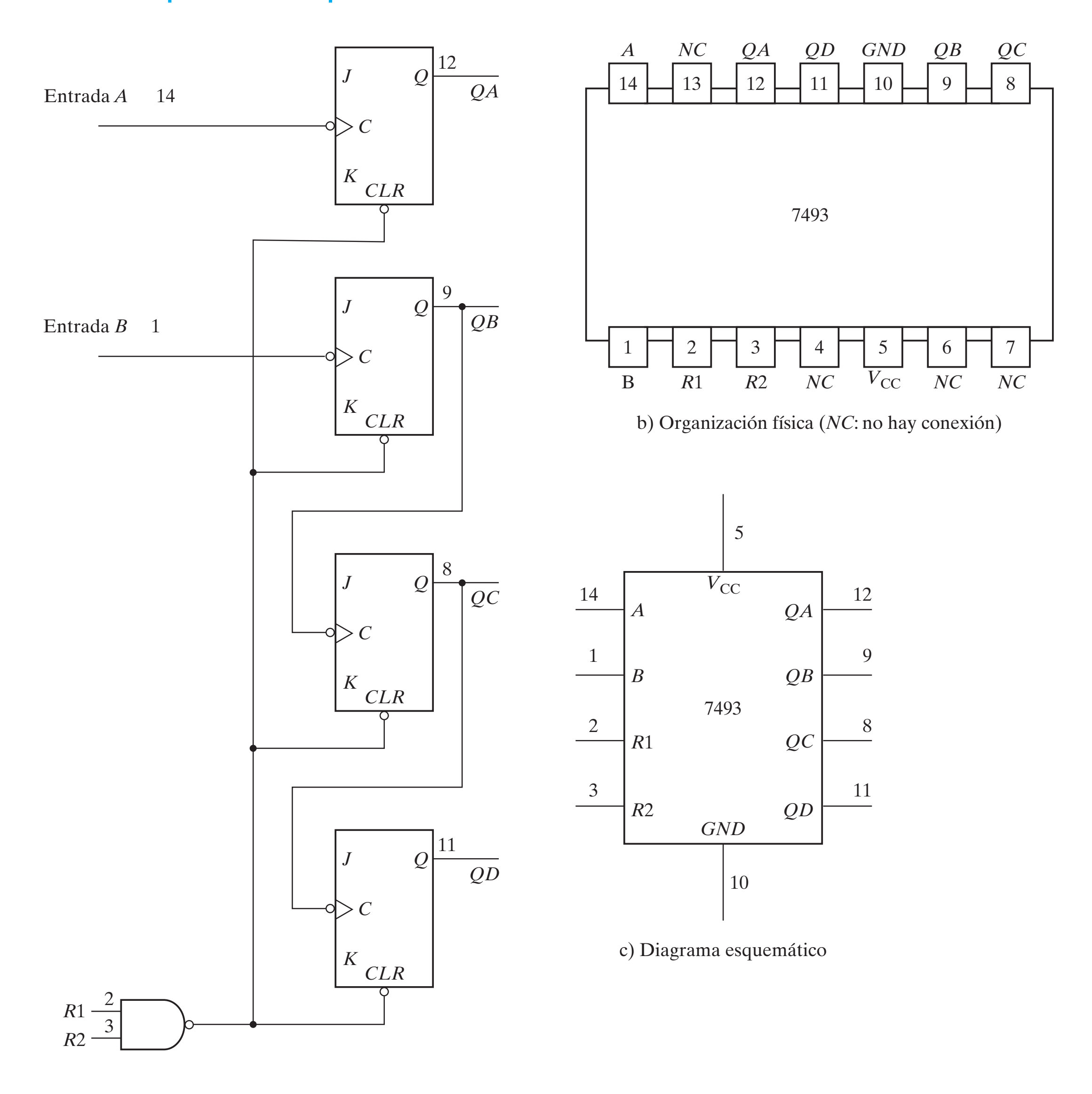


FIGURA 11-1
Compuertas digitales en paquetes de CI con números de identificación y asignación de terminales



a) Diagrama interno del circuito

b) Organización física (*NC*: no hay conexión)

c) Diagrama esquemático

FIGURA 11-2
CI tipo 7493: contador de rizo

El CI 7493 opera como contador de tres bits utilizando la entrada *B* y los flip-flops *QB*, *QC* y *QD*. Opera como contador de cuatro bits empleando la entrada *A* si la salida *QA* se conecta a la entrada *B*. Por tanto, para operar el circuito como contador de cuatro bits, es preciso tener una conexión externa entre la terminal 12 y la terminal 1. Las entradas de restablecimiento, *R*1 y *R*2 (terminales 2 y 3, respectivamente), deben aterrizarse. Las terminales 5 y 10 deben conectarse a una fuente de poder de 5 V. Los pulsos de entrada deben apli-

carse a la entrada *A* en la terminal 14, mientras que las cuatro salidas de flip-flop del contador se toman de *QA*, *QB*, *QC* y *QD* en las terminales 12, 9, 8 y 11, respectivamente, siendo *QA* el bit menos significativo.

La figura 11-2c) ilustra el símbolo gráfico que se usará para todos los circuitos MSI en este capítulo. Sólo se mostrará un diagrama de bloques similar al que se presenta en esta figura, para cada CI. Los símbolos de las entradas y salidas del diagrama de bloques de CI serán los empleados en el libro de datos. El funcionamiento del circuito se explicará haciendo referencia a diagramas lógicos de capítulos anteriores, y se especificará con una tabla de verdad o una tabla de función.

En el capítulo 12 se presentarán otros símbolos gráficos que podrían usarse para representar CI. Se trata de símbolos gráficos estándar aprobados por el Instituto de Ingenieros en Electricidad y Electrónica que se presentan en la norma IEEE 91-1984. Los símbolos gráficos estándar para las compuertas SSI son rectangulares, como se observa en la figura 12-1. El símbolo gráfico estándar para el CI 7493 se muestra en la figura 12-13. Se puede usar este símbolo en vez del de la figura 11-2c). En el capítulo 12 se presentan los símbolos gráficos estándar de los otros CI que se requieren para realizar los experimentos. Serán útiles para elaborar diagramas esquemáticos de los circuitos lógicos si se prefieren símbolos estándar.

En la tabla 11-1 se da una lista de los CI que se necesitan para los experimentos, junto con los números de las figuras en las que se presentan en este capítulo y los de las figuras del capítulo 12 donde se utilizan los símbolos gráficos estándar equivalentes.

El resto del capítulo contiene 19 secciones. Las primeras 18 presentan 18 experimentos con hardware en los que hay que usar circuitos integrados digitales. La sección 11-9 delinea experimentos de simulación HDL que requieren un compilador de Verilog HDL y un simulador.

Tabla 11-1
Circuitos integrados que se requieren para los experimentos

		Símbolo gráfico	
Número de CI	**Descripción**	**En cap. 11**	**En cap. 12**
	Diversas compuertas	Fig. 11-1	Fig. 12-1
7447	Decodificador BCD a 7 segmentos	Fig. 11-8	
7474	Flip-flops duales tipo *D*	Fig. 11-13	Fig. 12-9b)
7476	Flip-flops duales tipo *JK*	Fig. 11-12	Fig. 12-9a)
7483	Sumador binario de 4 bits	Fig. 11-10	Fig. 12-2
7493	Contador de rizo de 4 bits	Fig. 11-2	Fig. 12-13
74151	Multiplexor 8 × 1	Fig. 11-9	Fig. 12-7a)
74155	Decodificador 3 × 8	Fig. 11-7	Fig. 12-6
74157	Multiplexores 2 × 1 cuádruples	Fig. 11-17	Fig. 12-7b)
74161	Contador sincrónico de 4 bits	Fig. 11-15	Fig. 12-14
74189	Memoria de acceso aleatorio 16 × 4	Fig. 11-18	Fig. 12-15
74194	Registro de desplazamiento bidireccional	Fig. 11-19	Fig. 12-12
74195	Registro de desplazamiento de 4 bits	Fig. 11-16	Fig. 12-11
7730	Display LED de 7 segmentos	Fig. 11-8	
72555	Temporizador (igual que 555)	Fig. 11-21	

11-1 NÚMEROS DECIMALES Y BINARIOS

Este experimento ilustra la sucesión de conteo de los números binarios y la representación BCD (decimal codificado en binario). Sirve como introducción a la tabla de pruebas empleada en el laboratorio y familiariza al estudiante con el osciloscopio de rayos catódicos. La sección 1-2, sobre números binarios, y la sección 1-7, sobre números BCD, contienen material que convendría consultar antes de realizar el experimento.

Conteo binario

El CI tipo 7493 consta de cuatro flip-flops, como se indica en la figura 11-2. Se les puede conectar para que cuenten en binario o en BCD. El CI se conecta para operar como contador binario alambrando las terminales externas, como se muestra en la figura 11-3. Se conecta un cable entre la terminal 12 (salida *QA*) y la terminal 1 (entrada *B*). La entrada *A* (terminal 14) se conecta a un pulsador que genera pulsos individuales. Las dos entradas de restablecimiento, *R*1 y *R*2, se conectan a tierra. Las cuatro salidas se conectan a cuatro lámparas indicadoras, conectando el bit de orden bajo del contador (*QA*) a la lámpara de extrema derecha. No olvide alimentar 5 V y tierra al CI. Todas las conexiones deben realizarse con la fuente de poder en la posición de apagado.

Encienda la fuente de poder y observe las cuatro lámparas. El número de cuatro bits de la salida se incrementa en uno con cada pulso generado al accionarse el botón del pulsador. La cuenta llega al 15 binario y luego regresa a 0. Desconecte del pulsador la entrada del contador (terminal 14) y conéctela a un generador de reloj que produzca un tren de pulsos de baja frecuencia, aproximadamente un pulso por segundo. El conteo binario ahora será automático. Usaremos el contador binario en experimentos posteriores para generar señales binarias de entrada con las cuales probar circuitos combinacionales.

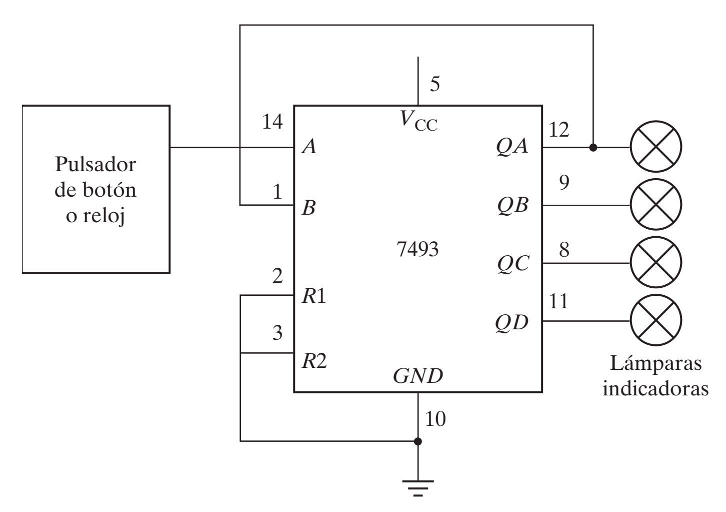


FIGURA 11-3
Contador binario

Exhibición en osciloscopio

Incremente la frecuencia del reloj a 10 kHz o más y conecte su salida a un osciloscopio. Observe la salida del reloj en el osciloscopio y dibuje su forma de onda. Utilizando un osciloscopio de rastreo dual, conecte la salida de *QA* a un canal y la salida del reloj al segundo canal. Observe que la salida de *QA* se complementa cada vez que el pulso de reloj efectúa una transición negativa de 1 a 0. También, la frecuencia de reloj en la salida del primer flip-flop es la mitad que la frecuencia de entrada del reloj. Cada flip-flop subsiguiente reduce a la mitad la frecuencia que recibe. El contador de 4 bits divide la frecuencia de entrada entre 16 en la salida *QD*. Genere un diagrama de temporización que muestre la relación de tiempo entre el reloj y las cuatro salidas del contador. Incluya por lo menos 16 ciclos de reloj. El procedimiento con un osciloscopio de rastreo dual es el siguiente. Primero, observe los pulsos de reloj y *QA* y registre sus formas de onda de temporización. Luego observe y registre las formas de onda de *QA* y de *QB*, luego las de *QB* con *QC* y, por último, las de *QC* con *QD*. El resultado final deberá ser un diagrama que indique la relación de tiempo entre el reloj y las cuatro entradas en un solo diagrama sinóptico que abarque por lo menos 16 ciclos de reloj.

Conteo BCD

La representación BCD emplea los números binarios del 0000 al 1001 para codificar los dígitos decimales del 0 al 9. El CI tipo 7493 puede operarse como contador BCD efectuando las conexiones externas señaladas en la figura 11-4. Las salidas *QB* y *QD* se conectan a las dos entradas de restablecimiento, *R*1 y *R*2. Cuando ambas entradas son 1, las cuatro celdas del contador se ponen en 0 independientemente del pulso de entrada. El contador parte de 0, y cada pulso de entrada lo incrementa en uno hasta llegar a la cuenta 1001. El siguiente pulso cambia

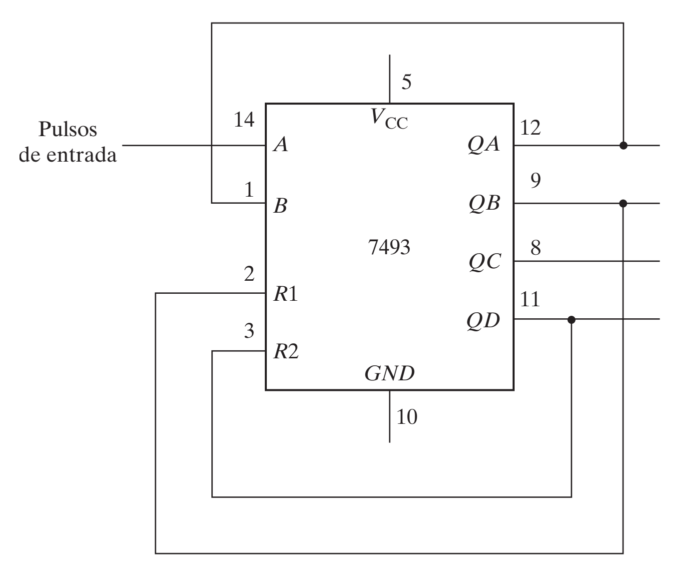


FIGURA 11-4
Contador BCD

la salida a 1010, lo que hace a *QB* y *QD* iguales a 1. Esta salida momentánea no se sostiene, porque las cuatro celdas inmediatamente se ponen en 0, así que la salida cambia a 0000. Así pues, el pulso que sigue al conteo 1001 cambia la salida a 0000, lo que corresponde a un conteo BCD.

Conecte el CI de modo que opere como contador BCD. Conecte la entrada a un pulsador, y las cuatro salidas, a lámparas indicadoras. Compruebe que el conteo vaya de 0000 hasta 1001.

Desconecte la entrada del pulsador y conéctela a un generador de reloj. Observe la forma de onda del reloj y de las cuatro salidas en el osciloscopio. Obtenga un diagrama de temporización exacto que muestre la relación de tiempos entre el reloj y las cuatro salidas. Incluya al menos 10 ciclos de reloj en la pantalla del osciloscopio y en el diagrama de temporización compuesto.

Patrón de salida

Si los pulsos de conteo que se alimentan al contador BCD son continuos, el contador sigue repitiendo la sucesión de 0000 a 1001 y de vuelta a 0000. Esto implica que cada bit de las cuatro salidas produce un patrón fijo de unos y ceros, que se repite cada 10 pulsos. Es posible predecir dichos patrones a partir de la lista de números binarios de 0000 a 1001. La lista muestra que la salida *QA*, al ser el bit menos significativo, produce un patrón alternado de unos y ceros. La salida *QD*, al ser el bit más significativo, produce un patrón de ocho ceros seguidos de dos unos. Obtenga el patrón de las otras dos salidas y verifique los cuatro patrones en el osciloscopio. Esto se hace con un osciloscopio de rastreo dual, exhibiendo los pulsos de reloj en un canal y una de las formas de onda de salida en el otro canal. El patrón de unos y ceros de la salida correspondiente se obtiene observando los niveles de salida en las posiciones verticales en las que los pulsos cambian de 1 a 0.

Otros conteos

El CI tipo 7493 puede conectarse de modo que cuente desde 0 hasta diversos conteos finales. Esto se hace conectando una o dos salidas a las entradas de restablecimiento *R*1 y *R*2. Por ejemplo, si *R*1 se conecta a *QA* en vez de *QB* en la figura 11-4, el conteo será de 0000 a 1000, o sea, uno menos que 1001 ($QD = 1$ y $QA = 1$).

Con base en lo que sabe acerca de la forma en que *R*1 y *R*2 afectan al conteo final, conecte el CI 7493 de modo que cuente desde 0000 hasta las cuentas finales siguientes:

a) 0101

b) 0111

c) 1011

Conecte cada circuito y verifique su sucesión de conteo aplicando pulsos con el pulsador y observando el conteo de salida en las lámparas indicadoras. Si el conteo inicial es un valor mayor que el conteo final, siga aplicando pulsos de entrada hasta que la salida se ponga en ceros.

11-2 COMPUERTAS LÓGICAS DIGITALES

En este experimento investigaremos el comportamiento lógico de diversas compuertas de CI:

7400 Compuertas NAND cuádruples de 2 entradas
7402 Compuertas NOR cuádruples de 2 entradas
7404 Inversores séxtuples
7408 Compuertas AND cuádruples de 2 entradas
7432 Compuertas OR cuádruples de 2 entradas
7486 Compuertas XOR cuádruples de 2 entradas

Las asignaciones de terminales de las diversas compuertas se señalan en la figura 11-1. “Cuádruple” significa que hay cuatro compuertas en el paquete. Las compuertas de lógica digital y sus características se explican en la sección 2-8. La implementación NAND se explica en la sección 3-6.

Tablas de verdad

Utilice una compuerta de cada CI de la lista anterior y obtenga la tabla de verdad para la compuerta. La tabla se obtiene conectando las entradas de la compuerta a interruptores, y la salida, a una lámpara indicadora. Compare sus resultados con las tablas de verdad de la figura 2-5.

Formas de onda

Para cada compuerta de la lista anterior, obtenga la relación de forma de onda entrada-salida. Las formas de onda se observan en el osciloscopio. Utilice las dos salidas de orden bajo de un contador binario (figura 11-3) para alimentar las entradas de la compuerta. Por ejemplo, en la figura 11-5 se aprecian el circuito y las formas de onda para la compuerta NAND. La pantalla del osciloscopio repetirá esta forma de onda, pero sólo hay que registrar la porción no repetitiva.

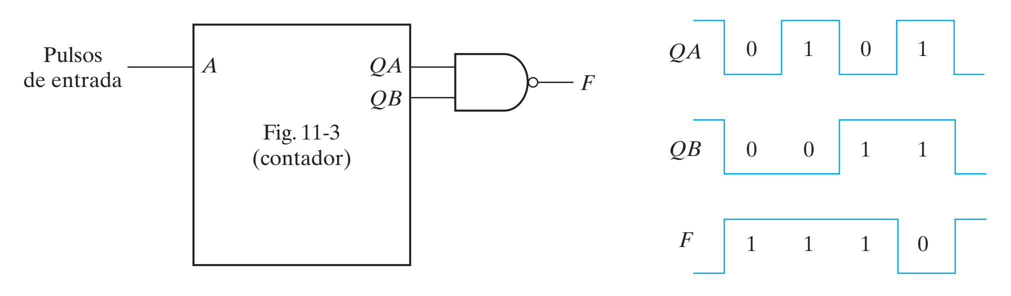


FIGURA 11-5
Formas de onda de la compuerta NAND

Retardo de propagación

Conecte en cascada los seis inversores del CI 7404. La salida será igual a la entrada, sólo que se retrasará por el tiempo que la señal tarda en propagarse por los seis inversores. Aplique pulsos de reloj a la entrada del primer inversor. Con el osciloscopio, determine el retardo desde esa entrada hasta la salida del sexto inversor durante el ascenso y una vez más durante el descenso del pulso. Esto se hace con un osciloscopio de rastreo doble aplicando los pulsos de reloj de entrada a uno de los canales, y la salida del sexto inversor, al otro. El control de base de tiempo se debe ajustar a la posición de tiempo por división más baja. El tiempo de ascenso o descenso de los dos pulsos deberá aparecer en la pantalla. Divida el retardo total entre 6 para obtener un retardo de propagación medio por inversor.

Compuerta NAND universal

Utilizando un solo CI 7400, conecte un circuito que produzca

a) un inversor
b) un AND de dos entradas
c) un OR de dos entradas
d) un NOR de dos entradas
e) un XOR de dos entradas (véase la figura 3-32)

En cada caso, verifique el circuito preparando su tabla de verdad.

Circuito NAND

Utilizando un solo CI 7400, construya un circuito con compuertas NAND que implemente la función booleana

$$F = AB + CD$$

1. Dibuje el diagrama de circuito.
2. Prepare la tabla de verdad de F en función de las cuatro entradas.
3. Conecte el circuito y verifique la tabla de verdad.
4. Registre los patrones de unos y ceros de F a medida que las entradas A, B, C y D cambian de 0 binario a 15 binario.
5. Conecte las cuatro salidas del contador binario de la figura 11-3 a las cuatro entradas del circuito NAND. Conecte a un canal de un osciloscopio de rastreo dual los pulsos de reloj que entran al contador, y al otro canal, la salida F. Observe y registre el patrón de unos y ceros de F después de cada pulso de reloj y compárelo con el patrón registrado en el paso 4.

11-3 SIMPLIFICACIÓN DE FUNCIONES BOOLEANAS

Este experimento ilustra la relación entre una función booleana y el diagrama lógico correspondiente. Las funciones booleanas se simplifican con el método de mapa, descrito en el capítulo 3. Los diagramas lógicos se dibujan empleando compuertas NAND, como se explica en la sección 3-6.

Los CI de compuertas que se usarán en los diagramas lógicos deben ser los de la figura 11-1 que contienen las compuertas NAND siguientes:

7400 NAND de 2 entradas

7404 Inversor (NAND de 1 entrada)

7410 NAND de 3 entradas

7420 NAND de 4 entradas

Si no se usa una entrada de una compuerta NAND, no debe dejarse abierta. Debe conectarse a otra entrada que sí se use. Por ejemplo, si el circuito necesita un inversor y hay una compuerta extra de dos entradas disponible en un CI 7400, ambas entradas deberán conectarse entre sí para formar la entrada única de un inversor.

Diagrama lógico

Esta parte del experimento parte de un diagrama lógico dado al que aplicaremos procedimientos de simplificación para reducir el número de compuertas y posiblemente el número de circuitos integrados. El diagrama lógico de la figura 11-6 requiere dos CI, un 7400 y un 7410. Los inversores para las entradas *x*, *y* y *z* se obtienen de las tres compuertas restantes del CI 7400. Si los inversores se tomaran de un CI 7404, el circuito habría requerido tres CI. Cabe señalar también que, al dibujar circuitos SSI, las compuertas no se encierran en bloques como se hace con los circuitos MSI.

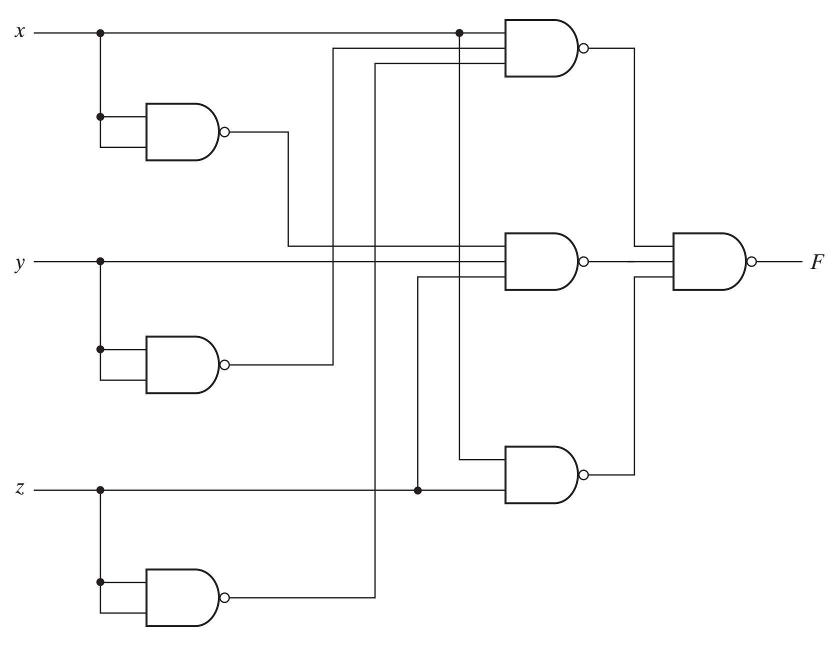


FIGURA 11-6
Diagrama lógico para el experimento 3

Asigne números de terminal a todas las entradas y salidas de las compuertas y conecte el circuito de modo que las entradas x, y y z se tomen de tres interruptores y la salida F vaya a una lámpara indicadora. Pruebe el circuito obteniendo su tabla de verdad.

Obtenga la función booleana del circuito y simplifíquela con el método de mapa. Construya el circuito simplificado sin desconectar el circuito original. Pruebe ambos circuitos aplicando entradas idénticas a los dos y observe las salidas. Demuestre que para cada una de las ocho posibles combinaciones de entrada, los dos circuitos tienen idéntica salida. Esto demostrará que el circuito simplificado se comporta exactamente igual que el original.

Funciones booleanas

Dadas las dos funciones booleanas en suma de minitérminos:

$$F_1(A, B, C, D) = (0, 1, 4, 5, 8, 9, 10, 12, 13)$$

$$F_2(A, B, C, D) = (3, 5, 7, 8, 10, 11, 13, 15)$$

simplifique las dos funciones con la ayuda de mapas. Obtenga un diagrama lógico compuesto con cuatro entradas, A, B, C y D, y dos salidas, F_1 y F_2. Implemente juntas las dos funciones utilizando un mínimo de circuitos integrados NAND. No duplique una compuerta si el término correspondiente se requiere para ambas funciones. En la medida de lo posible, utilice las compuertas sobrantes de los CI existentes, si las hay, como inversores. Conecte el circuito y verifique su operación. La tabla de verdad para F_1 y F_2 obtenida del circuito deberá coincidir con los minitérminos numerados.

Complemento

Grafique esta función booleana en un mapa:

$$F = A'D + BD + B'C + AB'D$$

Combine los unos del mapa para obtener la función simplificada F en suma de productos. Luego combine los ceros para obtener la función simplificada F' también en suma de productos. Implemente tanto F como F' utilizando compuertas NAND y conecte los dos circuitos a los mismos interruptores de entrada, pero a distintas lámparas indicadoras de salida. Obtenga la tabla de verdad de cada circuito en el laboratorio y demuestre que una es el complemento de la otra.

11-4 CIRCUITOS COMBINACIONALES

En este experimento diseñará, construirá y probará cuatro circuitos de lógica combinacional. Los dos primeros se construirán con compuertas NAND, el tercero con compuertas XOR, y el cuarto, con un decodificador y compuertas NAND. Los generadores de paridad se explican en la sección 3.8. La implementación con un decodificador se explica en la sección 4-8.

Ejemplo de diseño

Diseñe un circuito combinacional con cuatro entradas —*A*, *B*, *C* y *D*— y una salida, *F*. *F* debe ser 1 cuando $A = 1$, siempre que $B = 0$, o cuando $B = 1$, siempre que *C*, o bien, *D* también sea 1. De lo contrario, la salida debe ser 0.

1. Obtenga la tabla de verdad del circuito.
2. Simplifique la función de salida.
3. Dibuje el diagrama lógico del circuito empleando compuertas NAND con un mínimo de circuitos integrados.
4. Construya el circuito y pruebe que opere correctamente verificando las condiciones especificadas.

Lógica mayoritaria

Una lógica mayoritaria es un circuito digital cuya salida es 1 si la mayoría de las entradas es 1. En caso contrario, la salida es 0. Diseñe y pruebe un circuito mayoritario de tres entradas utilizando compuertas NAND con el mínimo de circuitos integrados.

Generador de paridad

Diseñe, construya y pruebe un circuito que genere un bit de paridad par a partir de cuatro bits de mensaje. Use compuertas XOR. Añada una compuerta XOR más para ampliar el circuito de modo que también genere un bit de paridad impar.

Implementación con un decodificador

Un circuito combinacional tiene tres entradas —*x*, *y* y *z*— y tres salidas —$F1$, F_2 y F_3. Las funciones booleanas simplificadas para el circuito son

$$F_1 = xz + x'y'z'$$

$$F_2 = x'y + xy'z'$$

$$F_3 = xy + x'y'z$$

Implemente y pruebe el circuito combinacional utilizando un CI decodificador 74155 y compuertas NAND externas.

El diagrama de bloques del decodificador y su tabla de verdad se reproducen en la figura 11-7. El 74155 puede conectarse como decodificador dual 2×4 o como decodificador sencillo 3×8. Si se desea un decodificador 3×8, las entradas *C*1 y *C*2 deben conectarse entre sí, lo mismo que las entradas *G*1 y *G*2, como se indica en el diagrama de bloques. La función del circuito es similar a la que se presenta en la figura 4-18. *G* es la entrada de habilitación y debe ser 0 para que el funcionamiento sea correcto. Las ocho entradas se rotulan con símbolos dados en el libro de datos. El 74155 utiliza compuertas NAND, y por ello la salida seleccionada pasa a 0 mientras todas las demás salidas permanecen en 1. La implementación con el decodificador es la que se muestra en la figura 4-21, excepto que las compuertas OR deben reemplazarse por compuertas externas NAND cuando se usa el 74155.

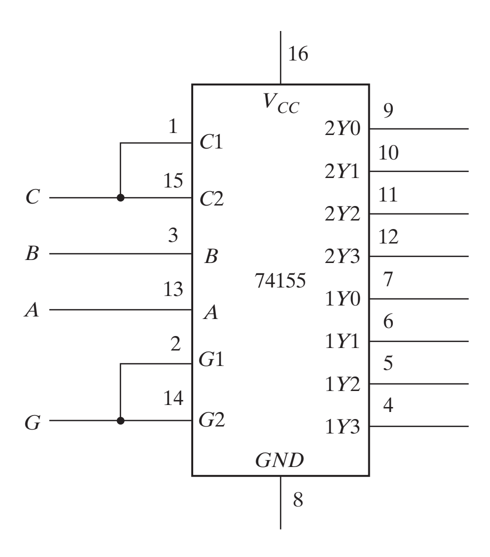


Tabla de verdad

Entradas				Salidas							
G	*C*	*B*	*A*	2*Y*0	2*Y*1	2*Y*2	2*Y*3	1*Y*0	1*Y*1	1*Y*2	1*Y*3
1	*X*	*X*	*X*	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1
0	0	1	0	1	1	0	1	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1
0	1	0	0	1	1	1	1	0	1	1	1
0	1	0	1	1	1	1	1	1	0	1	1
0	1	1	0	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	0

FIGURA 11-7
CI tipo 74155 conectado como decodificador 3 × 8

11-5 CONVERTIDORES DE CÓDIGO

La conversión de un código binario a otro es común en sistemas digitales. En este experimento diseñaremos y construiremos tres circuitos combinacionales convertidores. La conversión de códigos se trata en la sección 4-3.

Código Gray a binario

Diseñe un circuito combinacional con cuatro entradas y cuatro salidas que convierta un número de cuatro bits en código Gray (tabla 1-6) en el número binario de cuatro bits equivalente. Implemente el circuito con compuertas OR exclusivo. (Esto puede hacerse con un CI 7486.) Conecte el circuito a cuatro interruptores y cuatro lámparas indicadoras y verifique que funcione correctamente.

Complementador a nueve

Diseñe un circuito combinacional con cuatro líneas de entrada, las cuales representan un dígito decimal en BCD, y cuatro líneas de salida que generan el complemento a nueve del dígito de entrada. Incluya una quinta salida que detecte errores en el número BCD de entrada. Esta salida deberá ser 1 lógico si las cuatro entradas tienen una de las combinaciones que no se usan en el código BCD. Utilice cualquiera de las compuertas de la figura 11-1, pero use el número mínimo posible de circuitos integrados.

Display de siete segmentos

Los indicadores de siete segmentos sirven para exhibir cualquiera de los dígitos decimales, de 0 a 9. Por lo regular, el dígito decimal está en BCD. Un decodificador de BCD a siete segmentos acepta un dígito decimal en BCD y genera el código correspondiente para siete segmentos. Esto se ilustra gráficamente en el problema 4-9.

La figura 11-8 muestra las conexiones necesarias entre el decodificador y el display. El CI 7447 es un decodificador/manejador de BCD a siete segmentos. Tiene cuatro entradas para el dígito BCD. La entrada *D* es la más significativa, y la *A*, la menos significativa. El dígito BCD de cuatro bits se convierte en un código de siete segmentos en las salidas *a*-*g*. Las salidas del 7447 se aplican a las entradas del display de siete segmentos 7730 (o su equivalente). Este CI contiene los siete segmentos LED (diodo emisor de luz) en la cara superior del paquete. La entrada de la terminal 14 es el ánodo común (*CA*) para todos los LED. Hay que conectar un resistor de 47 Ω a V_{CC} para alimentar la corriente debida a los segmentos LED seleccionados. Otros CI equivalentes de display de siete segmentos podrían tener terminales de ánodo adicionales que podrían requerir resistores de distinto valor.

Construya el circuito que se muestra en la figura 11-8. Aplique los dígitos BCD de cuatro bits a través de cuatro interruptores y observe el display decimal de 0 a 9. Las entradas 1010 a

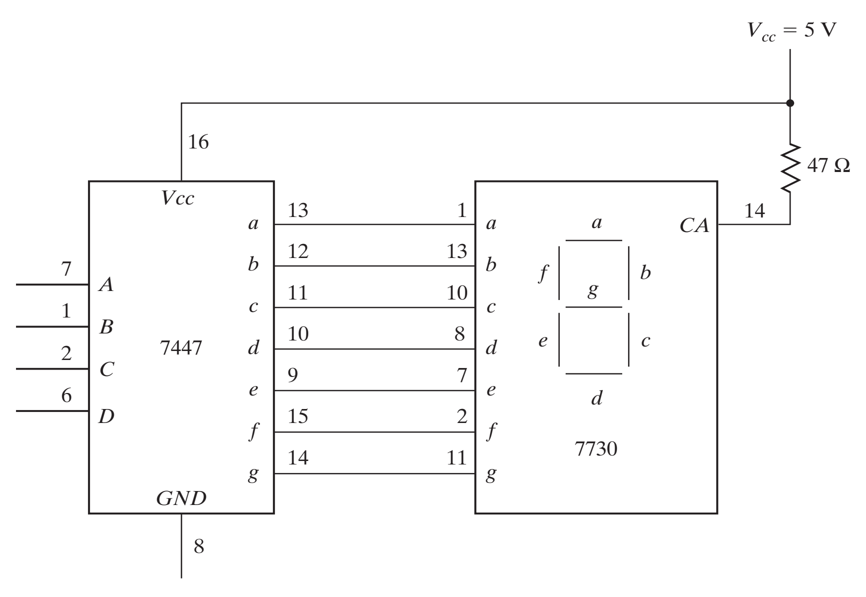


FIGURA 11-8
Decodificador de BCD a siete segmentos (7447) y display de siete segmentos (7730)

1111 carecen de significado en BCD. Dependiendo del decodificador, estos valores podrían poner en blanco el display o generar un patrón sin significado de segmentos. Observe y registre los patrones de salida que se exhiben con las seis combinaciones de entrada no utilizadas.

11-6 DISEÑO CON MULTIPLEXORES

En este experimento diseñará un circuito combinacional y lo implementará con multiplexores, como se explica en la sección 4-10. El multiplexor que se usará es el CI tipo 74151, que se observa en la figura 11-9. La construcción interna del 74151 es similar al diagrama de la fi-

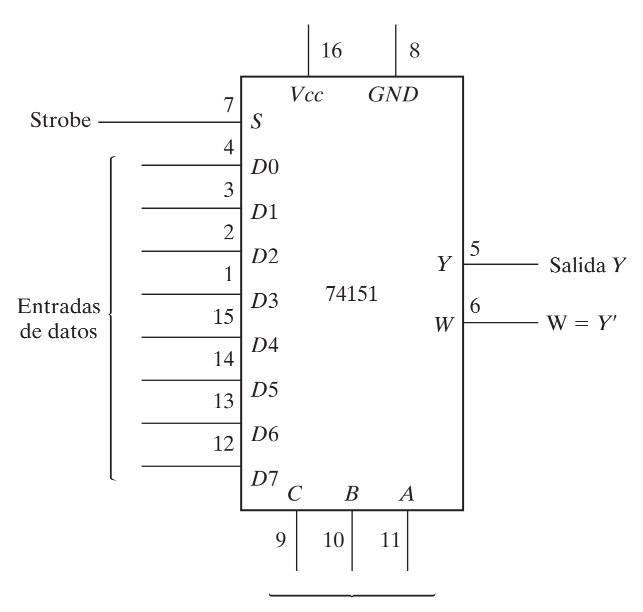


Tabla de función

Strobe S	Selección C	B	A	Salida Y
1	X	X	X	0
0	0	0	0	D0
0	0	0	1	D1
0	0	1	0	D2
0	0	1	1	D3
0	1	0	0	D4
0	1	0	1	D5
0	1	1	0	D6
0	1	1	1	D7

FIGURA 11-9
CI tipo 74151: multiplexor 8 × 1

gura 4-25, salvo que hay ocho entradas en lugar de cuatro. Las ocho entradas llevan los nombres *D*0 a *D*7. Las tres líneas de selección —*C*, *B* y *A*— seleccionan la entrada que se multiplexará y se aplicará a la salida. Un control estroboscópico *S* (*strobe*) actúa como señal de habilitación. La tabla de función especifica el valor de la salida *Y* en función de las líneas de selección. La salida *W* es el complemento de *Y*. Para que el circuito funcione correctamente, la entrada *strobe S* debe conectarse a tierra.

Especificaciones de diseño

Una corporación pequeña tiene 10 acciones, cada una de las cuales da a su titular derecho a un voto en las reuniones de accionistas. Las 10 acciones son propiedad de cuatro personas, a saber:

Sr. W: 1 acción

Sr. X: 2 acciones

Sr. Y: 3 acciones

Sra. Z: 4 acciones

Cada persona está provista de un interruptor que cierra al votar en favor y abre al votar en contra, según su participación accionaria.

Es necesario diseñar un circuito que exhiba el número total de acciones que votan en favor de cada propuesta. Utilice un display de siete segmentos y un decodificador, como se indica en la figura 11-8, para exhibir la cifra requerida. Si todas las acciones votan en contra de una propuesta, el display deberá estar en blanco. (Cabe señalar que si se alimenta la entrada binaria 15 al 7447, se apagan los siete segmentos.) Si 10 acciones votan en favor de una propuesta, el display deberá mostrar “0”. En los demás casos, el display mostrará un dígito decimal igual al número de acciones que votan en favor. Utilice cuatro multiplexores 74151 para diseñar el circuito combinacional que convierta las entradas generadas por los interruptores de los accionistas en el dígito BCD que se alimenta al 7447. No use 5 V para 1 lógico. Utilice la salida de un inversor cuya entrada está conectada a tierra.

11-7 SUMADORES Y RESTADORES

En este experimento construirá y probará diversos circuitos sumadores y restadores. Luego, usará el circuito restador para comparar la magnitud relativa de dos números. Los sumadores se tratan en la sección 4-3. La resta con complemento a dos se explica en la sección 1-6. En la figura 4-13 se muestra un sumador-restador paralelo de cuatro bits, y la comparación de dos números se explica en la sección 4-7.

Semisumador

Diseñe, construya y pruebe un circuito semisumador utilizando una compuerta XOR y dos compuertas NAND.

Sumador completo

Diseñe, construya y pruebe un circuito sumador completo utilizando dos CI, 7486 y 7400.

Sumador paralelo

El CI tipo 7483 es un sumador paralelo binario de cuatro bits. La asignación de terminales se indica en la figura 11-10. Los dos números binarios de 4 bits que constituyen las entradas son *A*1-*A*4 y *B*1-*B*4. La suma de cuatro bits se obtiene de *S*1-*S*4. *C*0 es el acarreo de entrada, y *C*4, el de salida.

Pruebe el sumador binario de cuatro bits 7483 conectando las terminales de fuente de poder y tierra. Luego conecte las cuatro entradas *A* a un número binario fijo, como 1001, y las cuatro entradas *B* y el acarreo de entrada a cinco interruptores de dos posiciones. Las cinco salidas se aplican a lámparas indicadoras. Efectúe la suma de unos cuantos números binarios y verifique que la suma y el acarreo de salida den los valores correctos. Demuestre que, si el acarreo de entrada es 1, suma 1 a la suma producida.

Sumador-restador

La resta de dos números binarios se efectúa obteniendo el complemento a dos del sustraendo y sumándolo al minuendo. El complemento a dos se obtiene tomando el complemento a uno y sumándole 1. Para efectuar $A - B$, se complementan los cuatro bits de *B*, se suman a los cuatro bits de *A* y se suma 1 a través del acarreo de entrada. Esto se hace como se muestra en la figura 11-11. Las cuatro compuertas XOR complementan los bits de *B* cuando el selector de modo $M = 1$ (porque $x \oplus 1 = x'$) y dejan los bits de *B* como están cuando $M = 0$ (porque $x \oplus 0 = x$). Así, cuando el selector de modo *M* es 1, el acarreo de entrada *C*0 es 1 y la suma producida es *A* más el complemento a dos de *B*. Cuando *M* es 0, el acarreo de entrada es 0 y la suma genera $A + B$.

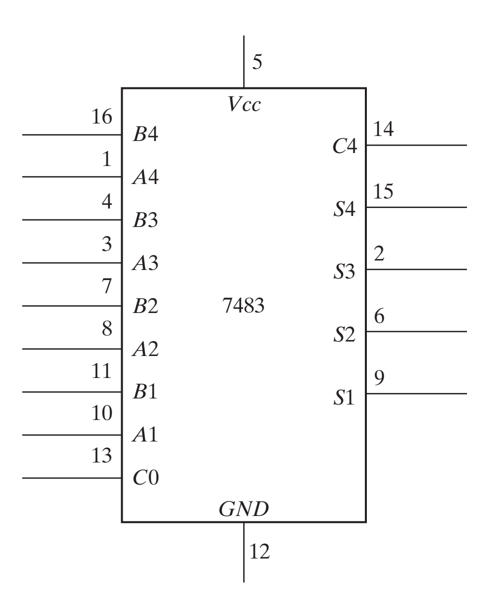


FIGURA 11-10
CI tipo 7483: sumador binario de cuatro bits

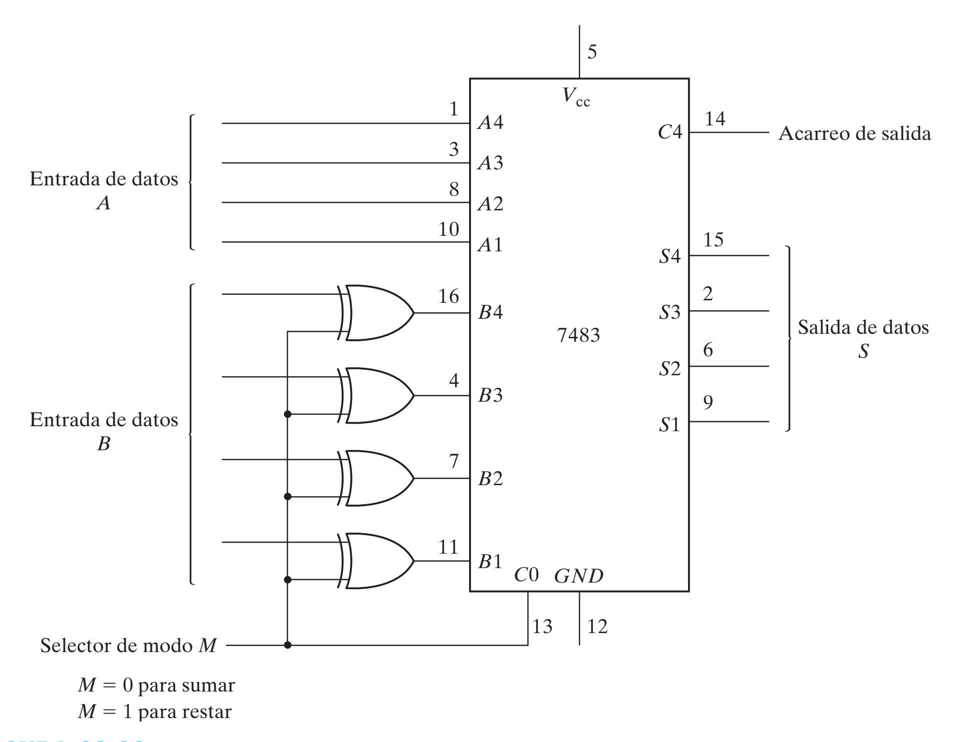


FIGURA 11-11
Sumador-restador de cuatro bits

Conecte el circuito sumador-restador y pruebe que funcione correctamente. Conecte las cuatro entradas A a un número binario fijo, 1001, y las entradas B a interruptores. Efectúe las operaciones siguientes y registre los valores de la suma producida y del acarreo de salida $C4$:

$9 + 5$	$9 - 5$
$9 + 9$	$9 - 9$
$9 + 15$	$9 - 15$

Compruebe que, al sumar, el acarreo de salida sea 1 cuando la suma exceda 15. Compruebe también que, si $A \geq B$, la operación de resta da la respuesta correcta, $A - B$, y el acarreo de salida $C4$ es 1. En cambio, cuando $A < B$, la resta da el complemento a dos de $B - A$ y el acarreo de salida es 0.

Comparador de magnitudes

La comparación de dos números es una operación que determina si un número es mayor o menor que otro, o igual a él. Dos números, A y B, se comparan restando primero $A - B$ como se hace en la figura 11-11. Si la salida en S es cero, sabremos que $A = B$. El acarreo de salida $C4$ determina la magnitud relativa: si $C4 = 1$, tenemos $A \geq B$; si $C4 = 0$, tenemos $A < B$; y cuando $C4 = 1$ y $S \neq 0$, tenemos $A > B$.

Es necesario añadir más elementos al circuito restador de la figura 11-1 para incluir la lógica de comparación. Esto se hace con un circuito combinacional de cinco entradas, $S1$-$S4$ y $C4$, y tres salidas denotadas por x, y y z, de modo que

$$x = 1 \quad \text{si } A = B \quad (S = 0000)$$
$$y = 1 \quad \text{si } A < B \quad (C4 = 0)$$
$$z = 1 \quad \text{si } A > B \quad (C4 = 1 \text{ y } S \neq 0000)$$

El circuito combinacional se puede implementar con dos CI, 7404 y 7408.

Construya el circuito comparador y pruebe su funcionamiento. Utilice por lo menos dos conjuntos de números para A y B al verificar las salidas x, y y z.

11-8 FLIP-FLOPS

Este experimento tiene por objetivo construir, probar e investigar el funcionamiento de diversos latches y flip-flops. La construcción interna de los latches y flip-flops se describe en las secciones 5-2 y 5-3.

Latch *SR*

Construya un latch *SR* con dos compuertas NAND acopladas en cruz. Conecte las dos entradas a interruptores, y las dos salidas, a lámparas indicadoras. Ponga los dos interruptores en 1 lógico y luego conmute momentáneamente cada interruptor, por separado, a la posición de 0 lógico y de vuelta a 1. Obtenga la tabla de función del circuito.

Latch *D*

Construya un latch *D* con cuatro compuertas NAND (sólo un CI 7400) y verifique su tabla de función.

Flip-flop amo-esclavo

Conecte un flip-flop *D* amo-esclavo utilizando dos latches *D* y un inversor. Conecte la entrada *D* a un interruptor, y la entrada de reloj, a un pulsador. Conecte la salida del latch amo a una lámpara indicadora, y la del latch esclavo, a otra lámpara. Ajuste el valor de la entrada de modo que sea el complemento de la salida. Oprima el botón del pulsador y luego suéltelo, para generar un solo pulso. Observe que el amo cambie cuando el pulso cambie a positivo y que el esclavo siga el cambio cuando el pulso cambie a negativo. Repita unas cuantas veces observando las dos lámparas indicadoras. Explique la sucesión de transferencia de la entrada al amo y del amo al esclavo.

Desconecte la entrada de reloj del pulsador y conéctela a un generador de reloj. Conecte a la entrada *D* la salida de complemento del flip-flop. Esto hará que el flip-flop se complemente con cada pulso de reloj. Utilizando un osciloscopio de doble rastreo, observe las formas de onda del reloj y de las salidas del amo y el esclavo. Verifique que el retardo entre las salidas del amo y el esclavo sea igual a la mitad positiva del ciclo de reloj. Prepare un diagrama de temporización que muestre la relación entre la forma de onda del reloj y las salidas del amo y el esclavo.

Flip-flop disparado por flanco

Construya un flip-flop tipo *D* disparado por flanco positivo utilizando seis compuertas NAND. Conecte la entrada de reloj a un pulsador, la entrada *D* a un interruptor de dos posiciones y la salida *Q* a una lámpara indicadora. Establezca el valor de *D* de modo que sea el complemento del valor de *Q*. Compruebe que la salida del flip-flop únicamente cambie en respuesta a una transición positiva del pulso de reloj. Verifique que la salida no cambie cuando la entrada del reloj es 1 lógico, cuando el reloj tiene una transición negativa ni cuando la entrada es 0 lógico. Siga cambiando la entrada *D* de modo que en todo momento corresponda a la salida *Q*.

Desconecte la entrada del pulsador y conéctela al generador de reloj. Conecte la salida de complemento a la entrada *D*. Esto hará que la salida se complemente con cada transición positiva del pulso de reloj. Utilizando un osciloscopio de rastreo dual, observe y registre la relación de temporización entre el reloj de entrada y la salida *Q*. Compruebe que la salida cambia en respuesta a una transición de borde positivo.

Flip-flops de CI

El CI tipo 7476 consta de dos flip-flops *JK* amo-esclavo con preestablecimiento y despeje. La asignación de terminales a cada flip-flop se muestra en la figura 11-12. La tabla de función especifica el funcionamiento del circuito. Las cuatro primeras columnas de la tabla especifican el funcionamiento de las entradas asincrónicas de preestablecimiento y despeje. Estas entradas

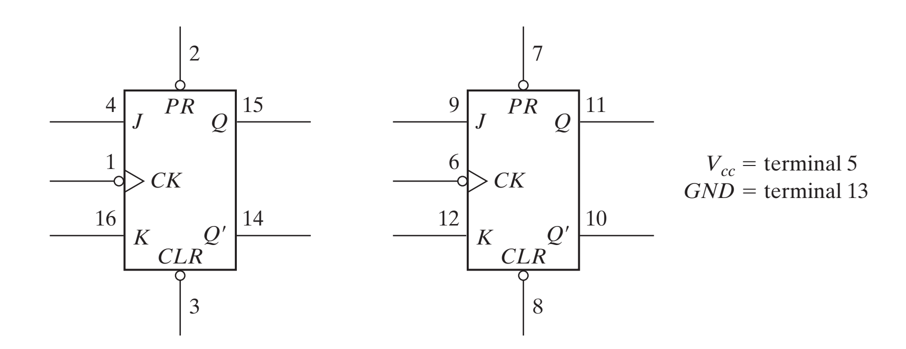


Tabla de función

Entradas					Salidas	
Preestab.	Despeje	Reloj	*J*	*K*	*Q*	*Q'*
0	1	*X*	*X*	*X*	1	0
1	0	*X*	*X*	*X*	0	1
0	0	*X*	*X*	*X*	1	1
1	1	⎍	0	0	Sin cambio	
1	1	⎍	0	1	0	1
1	1	⎍	1	0	1	0
1	1	⎍	1	1	Inversión	

FIGURA 11-12
CI tipo 7476: dos flip-flops JK amo-esclavo

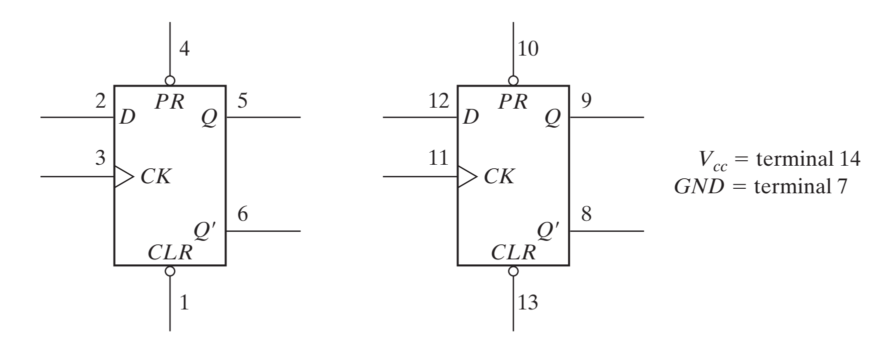


Tabla de función

Entradas				Salidas	
Preestab.	Despeje	Reloj	*D*	*Q*	*Q'*
0	1	*X*	*X*	1	0
1	0	*X*	*X*	0	1
0	0	*X*	*X*	1	1
1	1	↑	0	0	1
1	1	↑	1	1	0
1	1	0	*X*	Sin cambio	

FIGURA 11-13
CI tipo 7474: dos flip-flops *D* disparados por flanco positivo

se comportan como un latch *SR* NAND y son independientes del reloj y de las entradas *J* y *K* (las *X* indican condiciones de indiferencia). Las últimas cuatro columnas de la tabla de función especifican el funcionamiento con reloj cuando ambas entradas, preestablecimiento y despeje, se mantienen en 1 lógico. El valor de reloj se muestra como un solo pulso. La transición positiva del pulso hace que el flip-flop amo cambie, y la negativa, que cambie el flip-flop esclavo, así como la salida del circuito. Con $J = K = 0$, la salida no cambia. El flip-flop se invierte o complementa cuando $J = K = 1$. Investigue el funcionamiento de un flip-flop 7476 y verifique su tabla de función.

El CI tipo 7474 consiste en dos flip-flops *D* disparados por flanco positivo, con preestablecimiento y despeje. La asignación de terminales se indica en la figura 11-13. La tabla de función especifica las operaciones de preestablecimiento y despeje y el funcionamiento del reloj. Este último se señala con una flecha hacia arriba para indicar que se trata de un flip-flop disparado por (flanco) positivo. Investigue el funcionamiento de uno de los flip-flops y verifique su tabla de función.

11-9 CIRCUITOS SECUENCIALES

En este experimento se diseñarán, construirán y probarán tres circuitos secuenciales sincrónicos. Utilice el CI tipo 7476 (figura 11-12) o 7474 (figura 11-13). Escoja cualquier tipo de compuerta que reduzca al mínimo el número total de circuitos integrados. El diseño de circuitos secuenciales sincrónicos se trata en la sección 5-7.

Contador arriba-abajo con habilitación

Diseñe, construya y pruebe un contador de dos bits que cuente hacia arriba o hacia abajo. Una entrada de habilitación E determina si el contador está activo o inactivo. Si $E = 0$, el contador queda inhabilitado y conserva su cuenta actual aunque se apliquen pulsos de reloj a los flip-flops. Si $E = 1$, el contador se habilita y una segunda entrada, x, determina la dirección del conteo. Si $x = 1$, el circuito cuenta hacia arriba, siguiendo la sucesión 00, 01, 10, 11, después de lo cual se repite el conteo. Si $x = 0$, el circuito cuenta hacia abajo siguiendo la sucesión 11, 10, 01, 00, después de lo cual el conteo se repite. No utilice E para inhabilitar el reloj. Diseñe el circuito secuencial con E y x como entradas.

Diagrama de estados

Diseñe, construya y pruebe el circuito secuencial cuyo diagrama de estados se ilustra en la figura 11-14. Llame A y B a los dos flip-flops. La entrada será x, y la salida, y.

Conecte la salida del flip-flop menos significativo (B) a la entrada x y prediga la sucesión de estados y salida que se dará al aplicar pulsos de reloj. Verifique la transición de estados y la salida probando el circuito.

Diseño de un contador

Diseñe, construya y pruebe un contador que siga esta sucesión de estados binarios: 0, 1, 2, 3, 6, 7, 10, 11, 12, 13, 14, 15 y de vuelta a 0 para repetir. Los estados binarios 4, 5, 8 y 9 no se usan. El contador debe reiniciarse automáticamente; es decir, si el circuito es parte de cualquiera de los cuatro estados no válidos, los pulsos de conteo deberán hacer que el circuito pase a uno de los estados válidos para continuar el conteo correctamente.

Verifique que el circuito siga la sucesión de conteo requerida, y que se reinicie automáticamente. Esto se hace iniciando el circuito con cada uno de los estados no utilizados mediante las entradas de preestablecimiento y despeje, y aplicando luego pulsos para ver si el contador llega a uno de los estados válidos.

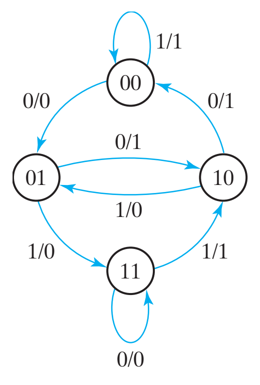


FIGURA 11-14
Diagrama de estados para el experimento 9

11-10 CONTADORES

En este experimento se construirán y probarán diversos circuitos contadores de rizo y sincrónicos. Los contadores de rizo se tratan en la sección 6-3, y los sincrónicos, en la sección 6-4.

Contador de rizo

Construya un contador binario de rizo de cuatro bits empleando dos CI 7476 (figura 11-12). Conecte a 1 lógico todas las entradas asincrónicas de preestablecimiento y despeje. Conecte la entrada de pulso de conteo a un pulsador y verifique que el contador funcione correctamente.

Modifique el contador de modo que cuente hacia abajo en lugar de hacia arriba. Compruebe que cada pulso de entrada decremente en 1 el contador.

Contador sincrónico

Construya un contador binario sincrónico de 4 bits y verifique su funcionamiento. Utilice dos CI 7476 y un CI 7408.

Contador decimal

Diseñe un contador BCD sincrónico que cuente de 0000 a 1001. Utilice dos CI 7476 y un CI 7408. Pruebe que la sucesión de conteo sea la correcta. Determine si el contador se reinicia automáticamente. Esto se hace iniciando el circuito con cada uno de los seis estados no utilizados mediante las entradas de preestablecimiento y despeje. Si el contador se reinicia automáticamente, la aplicación de pulsos deberá transferir el contador a uno de los estados válidos.

Contador binario con carga paralela

El CI tipo 74161 es un contador binario sincrónico de 4 bits con carga paralela y despeje asincrónico. Su lógica interna es similar a la del circuito de la figura 6-14. La asignación de terminales a las entradas y salidas se muestra en la figura 11-15. Cuando se habilita la señal de carga, las cuatro entradas de datos se transfieren a cuatro flip-flops internos, *QA* a *QD*, siendo *QD* el bit más significativo. Hay dos entradas que habilitan el conteo, *P* y *T*. Ambas deben ser 1 para que el contador funcione. La tabla de función es similar a la tabla 6-6 con una excepción: la entrada de carga del 74161 se habilita cuando es 0. Para cargar los datos de entrada, la entrada de despeje debe ser 1, y la de carga, 0. Las dos entradas de conteo tienen condiciones de indiferencia y podrían ser 1 o 0. Los flip-flops internos se disparan con la transición positiva del pulso de reloj. El circuito funciona como contador cuando la entrada de carga es 1 y ambas entradas de conteo, *P* y *T*, son 1. Si *P* o *T* cambian a 0, la salida no cambia. La salida de acarreo es 1 cuando las cuatro salidas de datos son 1. Efectúe un experimento para verificar que el funcionamiento del CI 74161 sea el indicado en la tabla de función.

Muestre cómo puede hacerse que el CI 74161, junto con una compuerta NAND de dos entradas, opere como contador BCD sincrónico que cuente de 0000 a 1001. No utilice la entrada de despeje. Utilice la compuerta NAND para detectar la cuenta 1001, lo que hará que se carguen ceros en el contador.

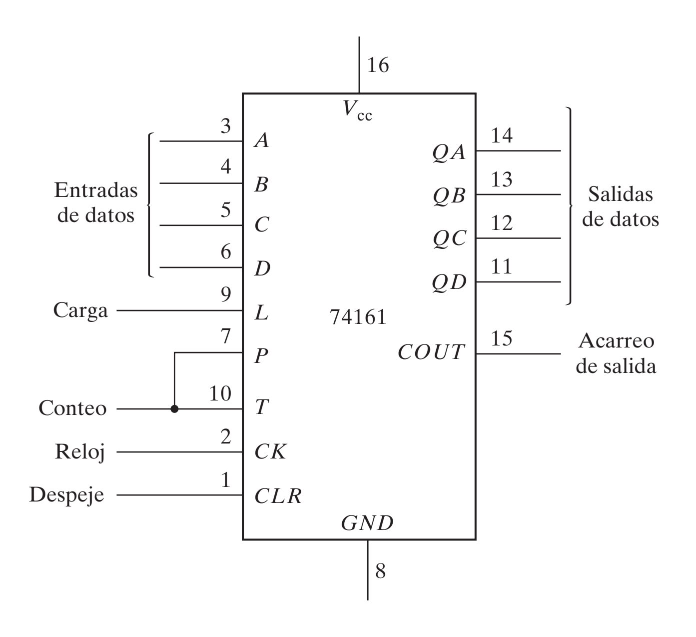


Tabla de función

Despeje	Reloj	Carga	Conteo	Función
0	*X*	*X*	*X*	Poner en 0 las salidas
1	↑	0	*X*	Cargar datos de entrada
1	↑	1	1	Contar al siguiente valor binario
1	↑	1	0	Ningún cambio en la salida

FIGURA 11-15
CI tipo 74161: contador binario con carga paralela

11-11 REGISTROS DE DESPLAZAMIENTO

En este experimento investigaremos el funcionamiento de los registros de desplazamiento. El CI empleado es el registro de desplazamiento 74195 con carga paralela. Los registros de desplazamiento se explican en la sección 6-2.

CI de registro de desplazamiento

El CI tipo 74195 es un registro de desplazamiento de cuatro bits con carga paralela y despeje asincrónico. La asignación de terminales a las entradas y salidas se indica en la figura 11-16. La línea de control única, rotulada SH/LD (*shift/load*, desplazar/cargar) determina el funcionamiento sincrónico del registro. Si $SH/LD = 0$, la entrada de control está en el modo de carga y las cuatro entradas de datos se transfieren a los cuatro flip-flops internos, *QA*-*QD*. Cuando $SH/LD = 1$, la entrada de control está en el modo de desplazamiento y la información contenida en el registro se desplaza hacia la derecha, desde *QA* hacia *QD*. La entrada en serie que

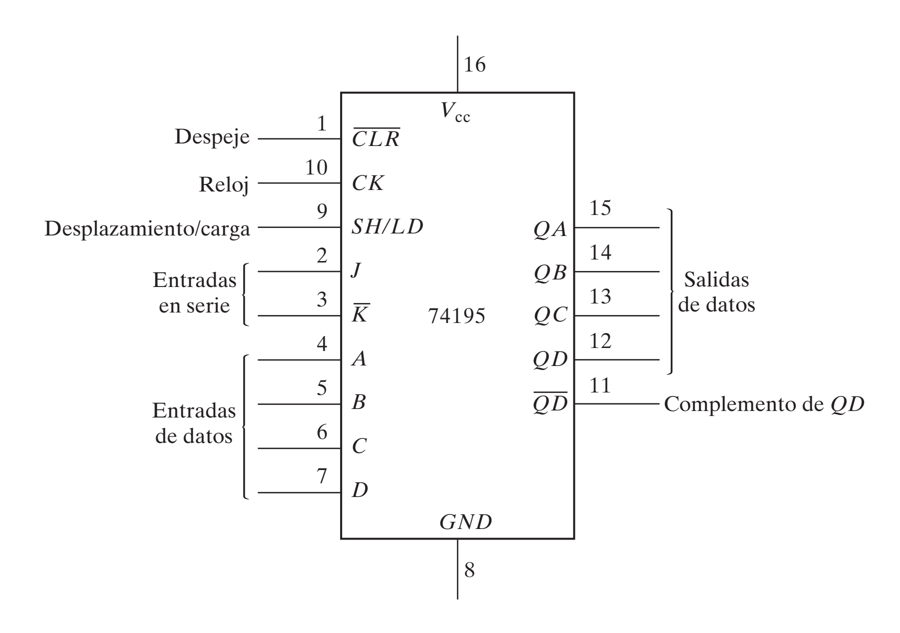


Tabla de función

Despeje	Desplaza-miento/carga	Reloj	J	$\overline{K}$	Entradas en serie	Función
0	X	X	X	X	X	Despeje asincrónico
1	X	0	X	X	X	Sin cambio en la salida
1	0	↑	X	X	X	Cargar datos de entrada
1	1	↑	0	0	0	Desplazar de QA hacia QD, $QA = 0$
1	1	↑	1	1	1	Desplazar de QA hacia QD, $QA = 1$

FIGURA 11-16
CI tipo 74195: registro de desplazamiento con carga paralela

llega a QA durante el desplazamiento se determina a partir de las entradas J y $\overline{K}$. Las dos entradas se comportan como la J y el complemento de la K de un flip-flop JK. Cuando tanto J como $\overline{K}$ son 0, el flip-flop QA se pone en 0 después del desplazamiento. Si ambas entradas son 1, QA se pone en 1 después del desplazamiento. Las otras dos condiciones de las entradas J y $\overline{K}$ hacen que la salida del flip-flop QA se complemente o no cambie después del desplazamiento.

La tabla de función del 74195 muestra el modo de operación del registro. Cuando la entrada de despeje cambia a 0, los cuatro flip-flops se ponen en cero asincrónicamente, es decir, sin necesidad del reloj. Las operaciones sincrónicas se efectúan con una transición positiva del reloj. Para cargar los datos de entrada, SH/LD debe ser 0 y debe haber una transición positiva de pulso de reloj. Para desplazar a la derecha, SH/LD debe ser 1. Las entradas J y $\overline{K}$ se deben conectar entre sí para formar la entrada en serie.

Efectúe un experimento que verifique el funcionamiento del CI 74195. Compruebe que efectúe todas las operaciones numeradas en la tabla de función. Incluya en su tabla de función las dos condiciones para $J\overline{K} = 01$ y 10.

Contador anular

Un contador anular es un registro de desplazamiento circular, en el que la señal de la salida en serie *QD* es la entrada en serie. Conecte las entradas *J* y $\overline{K}$ entre sí para formar la entrada en serie. Utilice la condición de carga para preestablecer el contador anular en un valor inicial de 1000. Haga que el bit dé vueltas con la condición de desplazamiento y verifique el estado del registro después de cada pulso de reloj.

Un contador anular con extremo conmutado utiliza la salida de complemento de *QD* como entrada en serie. Preestablezca el contador anular de extremo conmutado en 0000 y prediga la sucesión de estados que resulta del desplazamiento. Compruebe su predicción observando la sucesión de estados después de cada desplazamiento.

Registro de desplazamiento con retroalimentación

Un registro de desplazamiento con retroalimentación es un registro de desplazamiento cuya entrada en serie está conectada a alguna función de salidas selectas del registro. Conecte un registro de desplazamiento con retroalimentación cuya entrada en serie sea el OR exclusivo de las salidas *QC* y *QD*. Prediga la sucesión de estados partiendo del estado 1000. Verifique su predicción observando la sucesión de estados después de cada pulso de reloj.

Registro de desplazamiento bidireccional

El CI 74195 sólo puede desplazar a la derecha, de *QA* hacia *QD*. Es posible convertirlo en un registro de desplazamiento bidireccional utilizando el modo de carga para obtener una operación de desplazamiento a la izquierda (de *QD* hacia *QA*). Esto se logra conectando la salida de cada flip-flop a la entrada del flip-flop que está a su izquierda y utilizando el modo de carga de la entrada *SH*/*LD* como control de desplazamiento a la izquierda. La entrada *D* se convierte en la entrada en serie para la operación de desplazamiento a la izquierda.

Conecte el 74195 como registro de desplazamiento bidireccional (sin carga paralela). Conecte a un interruptor de dos posiciones la entrada en serie para desplazamiento a la derecha. Implemente el desplazamiento a la izquierda como contador anular conectando la salida en serie *QA* a la entrada en serie *D*. Despeje el registro y luego verifique su funcionamiento desplazando un solo 1 alimentado con el interruptor de la entrada en serie. Desplace a la derecha otras tres veces e inserte ceros con el interruptor de la entrada en serie. Luego desplace a la izquierda con el control de desplazamiento a la izquierda (carga). El 1 deberá seguir siendo visible durante el desplazamiento.

Registro de desplazamiento bidireccional con carga paralela

El CI 74195 se convierte en un registro de desplazamiento bidireccional con carga paralela combinándolo con un circuito multiplexor. Usaremos el CI tipo 74157 para este fin. Este CI contiene cuatro multiplexores de dos líneas a una cuya lógica interna se esquematiza en la figura 4-26. La asignación de terminales a las entradas y salidas del 74157 se muestra en la figura 11-17. Advierta que en el 74157 la entrada de habilitación se describe como *strobe*.

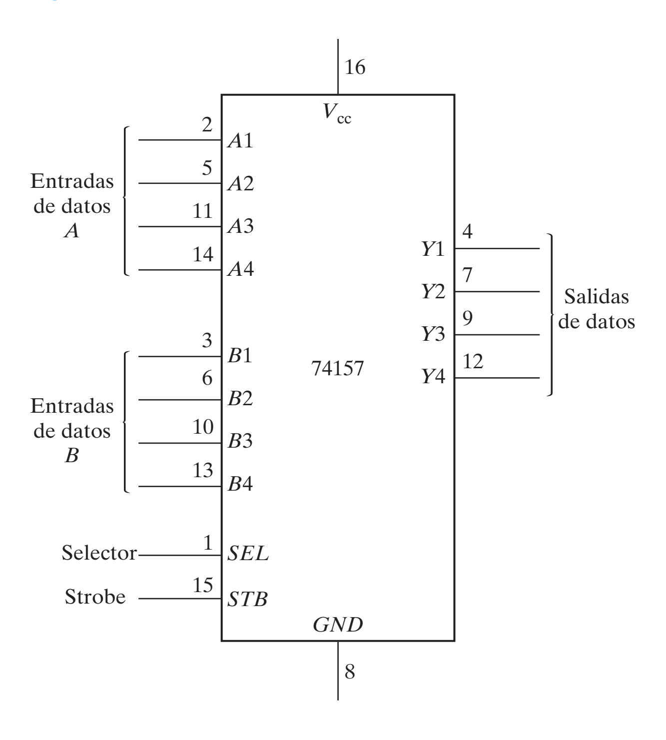


Tabla de función

Strobe	Selector	Salidas de datos *Y*
1	*X*	Puros ceros
0	0	Selecciona entradas de datos *A*
0	1	Selecciona entradas de datos *B*

FIGURA 11-17
CI tipo 74157: multiplexor cuádruple 2 × 1

Construya un registro de desplazamiento bidireccional con carga paralela utilizando el registro 74195 y el multiplexor 74157. El circuito deberá ser capaz de efectuar las operaciones siguientes:

1. Despeje asincrónico
2. Desplazar a la derecha
3. Desplazar a la izquierda
4. Carga paralela
5. Despeje sincrónico

Deduzca una tabla para las cinco operaciones en función de las entradas de despeje, reloj y *SH/LD* del 74195 y las entradas *strobe* y selector del 74157. Conecte el circuito y verifique la tabla de función. Utilice la condición de carga paralela para alimentar un valor inicial al registro y conecte las salidas en serie a las entradas en serie de ambos desplazamientos para no perder la información binaria durante el desplazamiento.

11-12 SUMA EN SERIE

En este experimento se construirá y probará un circuito sumador-restador en serie. La suma en serie de dos números binarios se efectúa con registros de desplazamiento y un sumador completo, como se explica en la sección 6-2.

Sumador en serie

Partiendo del diagrama de la figura 6-6, diseñe y construya un sumador en serie de 4 bits utilizando los circuitos integrados siguientes: 74195 (dos), 7408, 7486 y 7476. Incluya lo necesario para que el registro *B* acepte datos en paralelo de cuatro interruptores de dos posiciones, y conecte su entrada en serie a tierra para que se introduzcan ceros por la izquierda al registro *B* durante la suma. Incluya un interruptor de dos posiciones para despejar los registros y el flip-flop. Se necesitará otro interruptor para especificar si el registro *B* debe aceptar datos en paralelo o se debe desplazar durante la suma.

Prueba del sumador

Para probar su sumador en serie, efectúe la suma binaria $5 + 6 + 15 = 26$. Esto se hace despejando primero los registros y el flip-flop de acarreo. Cargue en paralelo el valor binario 0101 en el registro *B*. Aplique cuatro pulsos para sumar *B* y *A* en serie y compruebe que el resultado en *A* sea 0101. (Los pulsos de reloj para el 7476 deben ser como se indica en la figura 11-12.) Cargue en paralelo 0110 en *B* y súmelo a *A* en serie. Compruebe que *A* contenga la suma correcta. Cargue en paralelo 1111 en *B* y súmelo a *A*. Compruebe que el valor en *A* sea 1010 y que el flip-flop de acarreo sea 1.

Despeje los registros y el flip-flop y pruebe otros números para verificar que el sumador en serie esté funcionando correctamente.

Sumador-restador en serie

Si seguimos el procedimiento descrito en la sección 6-2 para el diseño de un restador en serie (que resta $A - B$), veremos que la diferencia de salida es igual a la suma de salida, pero que la entrada a *J* y *K* del flip-flop de acarreo necesita el complemento de *QD* (disponible en el 74195). Con las otras dos compuertas XOR del 7486, convierta el sumador en serie en un sumador-restador en serie con un control de modo *M*. Cuando $M = 0$, el circuito suma $A + B$. Cuando $M = 1$, el circuito resta $A - B$ y el flip-flop contiene el préstamo en vez del acarreo.

Pruebe la parte de sumador del circuito repitiendo las operaciones recomendadas en la subsección anterior, a fin de asegurarse de que la modificación no haya alterado el funcionamiento. Pruebe la parte de restador en serie efectuando las operaciones $15 - 4 - 5 - 13 = -7$. Se puede transferir 15 binario al registro *A* despejándolo primero y sumándole luego 15 que previamente se cargó en *B*. Verifique los resultados intermedios durante la resta. El -7 aparecerá como complemento a dos de 7 con préstamo 1 en el flip-flop.

11-13 UNIDAD DE MEMORIA

En este experimento se investigará el comportamiento de una unidad de memoria de acceso aleatorio (RAM) y su capacidad de almacenamiento. Se usará la RAM para simular una memoria de sólo lectura (ROM). Luego se utilizará el simulador de ROM para implementar circuitos combinacionales, como se explica en la sección 7-5. La unidad de memoria se trata en las secciones 7-2 y 7-3.

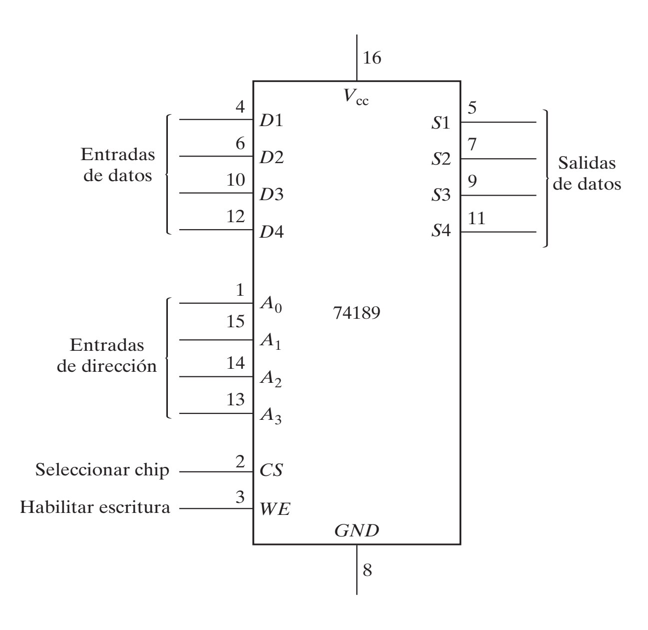


Tabla de función

CS	*WE*	Operación	Salidas de datos
0	0	Escribir	Alta impedancia
0	1	Leer	Complemento de palabra seleccionada
1	*X*	Inhabilitar	Alta impedancia

FIGURA 11-18
CI tipo 74189: RAM 16 × 4

CI de RAM

El CI tipo 74189 es una memoria de acceso aleatorio de 16 × 4. La lógica interna es similar al circuito de la figura 7-6, que es una RAM de 4 × 4. La asignación de terminales a las entradas y salidas se muestra en la figura 11-18. Las cuatro entradas de dirección seleccionan una de 16 palabras de la memoria. El bit menos significativo de la dirección es A_0, y el más significativo, A_3. La entrada de selección de chip (*CS*, *chip select*), debe ser 0 para habilitar la memoria. Si *CS* es 1, la memoria queda inhabilitada y las cuatro salidas están en el estado de alta impedancia. La entrada de habilitación de escritura (*WE*, *write enable*) determina el tipo de operación, como se indica en la tabla de función. Se efectúa una escritura cuando $WE = 0$. Ello consiste en una transferencia del número binario que está en las entradas de datos a la palabra seleccionada de la memoria. La operación de lectura se efectúa cuando $WE = 1$, y transfiere el valor de complemento almacenado en la palabra seleccionada, a las líneas de datos de salida. La memoria tiene salidas de tres estados para facilitar la expansión de memoria.

Prueba de la RAM

Puesto que las salidas del 74189 producen los valores de complemento, es preciso insertar cuatro inversores para cambiar las salidas a su valor normal. La RAM puede probarse después de efectuar las conexiones siguientes: conecte las entradas de dirección a un contador binario utilizando el CI 7493 (figura 11-3). Conecte las cuatro entradas de datos a interruptores de dos posiciones,

y las salidas de datos, a cuatro inversores 7404. Incluya cuatro lámparas indicadoras para la dirección y cuatro más para las salidas de los inversores. Conecte la entrada *CS* a tierra, y la *WE*, a un interruptor de dos posiciones (o a un pulsador que genere un pulso negativo). Almacene unas cuantas palabras en la memoria y luego léalas para verificar que las operaciones de escritura y lectura estén funcionando correctamente. Hay que tener cuidado al usar el interruptor *WE*. Mantenga la entrada *WE* en el modo de lectura continuamente, a menos que quiera escribir en la memoria. La forma correcta de escribir requiere colocar primero la dirección en el contador y luego las entradas en los cuatro interruptores de dos posiciones. Para almacenar la palabra en la memoria, cambie el interruptor *WE* a la posición de escritura y luego vuélvalo a la posición de lectura. Tenga cuidado de no modificar la dirección ni las entradas cuando *WE* esté en el modo de escritura.

Simulador de ROM

Se obtiene un simulador de ROM con una RAM operándola únicamente en el modo de lectura. El patrón de unos y ceros se introduce primero en la RAM simuladora colocando la unidad momentáneamente en el modo de escritura. Se efectúa la simulación colocando la unidad en el modo de lectura y tomando las líneas de dirección como entradas de la ROM. Entonces, la ROM podrá utilizarse para implementar cualquier circuito combinacional.

Implemente un circuito combinacional utilizando el simulador de ROM que convierte un número binario de 4 bits en su código Gray equivalente, definido en la tabla 1-6. Esto se hace como sigue. Obtenga la tabla de verdad del convertidor de código. Almacene la tabla de verdad en la memoria 74189 alimentando el valor binario a las entradas de dirección, y el valor correspondiente en código Gray, a las entradas de datos. Una vez escritas en la memoria las 16 filas de la tabla, se establece el simulador de ROM conectando permanentemente la línea *WE* a 1 lógico. Verifique el convertidor de código aplicando las entradas a las líneas de dirección y verificando que en las líneas de salida de datos aparezcan las salidas correctas.

Expansión de memoria

Expanda la unidad de memoria a una RAM de 32×4 utilizando dos CI 74189. Use las entradas *CS* para seleccionar el CI requerido. Dado que las salidas de datos son de tres estados, es posible conectar entre sí pares de terminales para obtener una operación de OR lógico entre los dos CI. Pruebe su circuito utilizándolo como simulador de ROM que sume un número de 3 bits a un número de 2 bits para producir una suma de 4 bits. Por ejemplo, si la entrada de la ROM es 10110, la salida deberá ser $101 + 10 = 0111$. (Los tres primeros bits de la entrada representan 5 binario, los últimos dos bits representan 2, y la suma producida es 7.) Utilice el contador para alimentar cuatro bits de la dirección y un interruptor para el quinto bit de la dirección.

11-14 FRONTÓN CON LÁMPARAS

En este experimento construiremos un juego electrónico de frontón utilizando una lámpara para simular la pelota en movimiento. Este proyecto ilustra la aplicación de un registro de desplazamiento bidireccional con carga paralela. También muestra el funcionamiento de las entradas asincrónicas de los flip-flops. Primero presentaremos un CI que se necesita para este experimento y luego mostraremos el diagrama lógico del juego de frontón simulado con lámparas.

CI tipo 74194

Éste es un registro de desplazamiento bidireccional con carga paralela. La lógica interna es similar a la de la figura 6-7. La asignación de terminales a las entradas y salidas se aprecia en la figura 11-19. Las dos entradas de control de modo determinan el tipo de operación, según especifica la tabla de función.

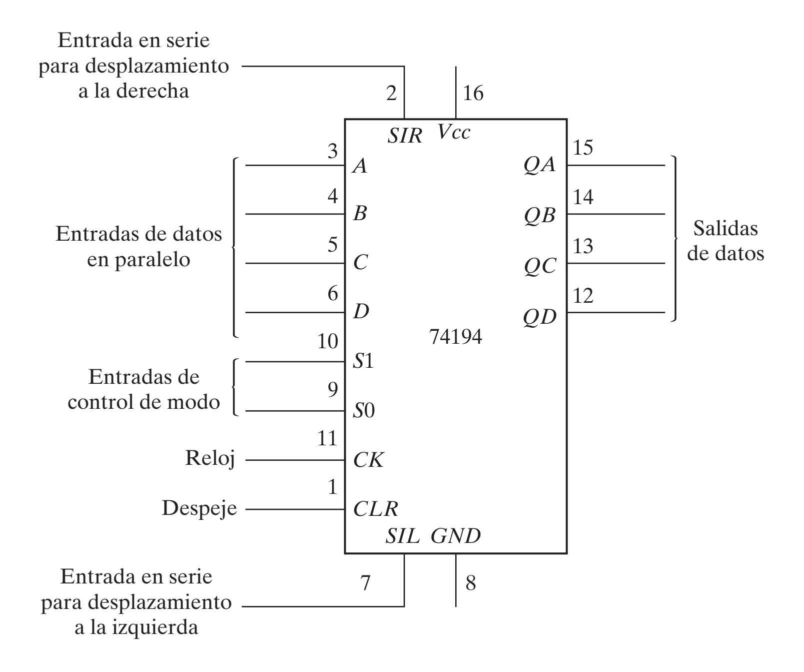


Tabla de función

Despeje	Reloj	Modo $S1$	$S0$	Función
0	X	X	X	Poner salidas en 0
1	↑	0	0	Ningún cambio en la salida
1	↑	0	1	Desplazamiento a la derecha de QA a QD. SIR a QA
1	↑	1	0	Desplazamiento a la izquierda de QD a QA. SIL a QD
1	↑	1	1	Carga paralela de datos de entrada

FIGURA 11-19
CI tipo 74194: registro de desplazamiento bidireccional con carga paralela

Diagrama lógico

El diagrama lógico del frontón electrónico con lámparas aparece en la figura 11-20. Consta de dos CI 74194, un CI 7474 de dos flip-flops *D* y tres CI de compuertas: 7400, 7404 y 7408. La pelota se simula con una luz en movimiento que se desplaza a la izquierda o a la derecha mediante el registro de desplazamiento bidireccional. La rapidez con que se mueve la luz depende de la frecuencia del reloj. Primero se inicia el circuito con el interruptor *restablecer*. El interruptor *inicio* pone en marcha el juego colocando la pelota (una lámpara indicadora) en la extrema derecha. Luego, el jugador debe oprimir el botón del pulsador para que la pelota comience a moverse hacia la izquierda. La luz se desplaza hacia la izquierda hasta llegar a la po-

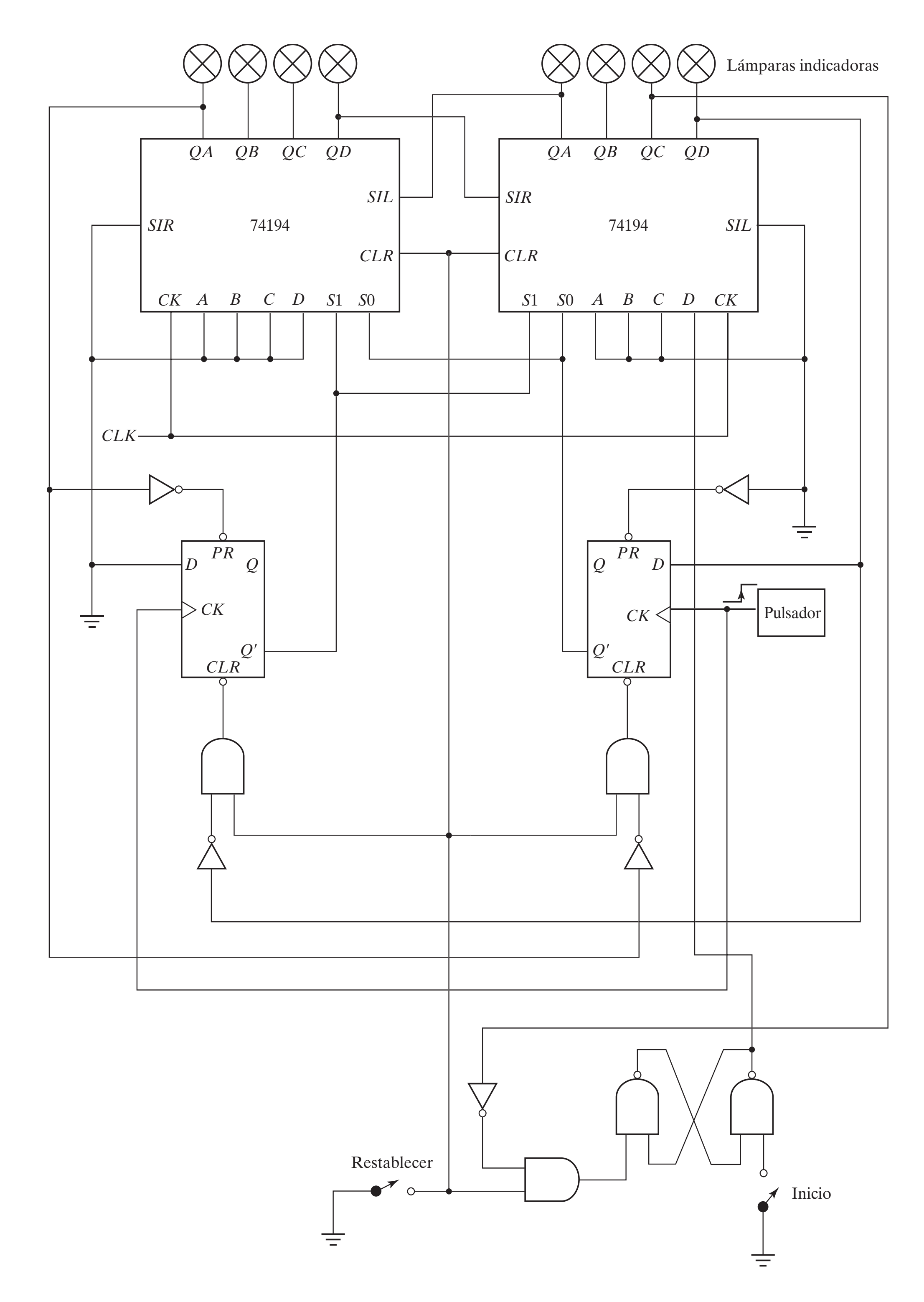


FIGURA 11-20
Diagrama lógico del frontón con lámparas

sición de extrema izquierda (la pared), para después regresar hacia el jugador invirtiendo la dirección del desplazamiento. Una vez que la luz ha llegado a la posición de extrema derecha, el jugador deberá oprimir otra vez el botón para invertir la dirección del desplazamiento. Si el jugador acciona el pulsador antes de tiempo o se tarda demasiado, la "pelota" desaparece (la luz se apaga). Se puede reiniciar el juego cerrando y abriendo el interruptor de inicio. Durante el juego, este interruptor debe estar abierto (1 lógico).

Análisis del circuito

Antes de conectar el circuito, analice el diagrama lógico hasta tener la certeza de que le queda claro el funcionamiento del circuito. En particular, trate de contestar estas preguntas:

1. ¿Qué función tiene el interruptor *restablecer*?
2. Explique cómo se enciende la lámpara de la extrema derecha cuando se aterriza el interruptor *inicio*. ¿Por qué es necesario colocar ese interruptor en la posición de 1 lógico antes de que inicie el juego?
3. ¿Qué sucede con las dos entradas de control de modo, $S1$ y $S0$, una vez que se pone en movimiento la pelota?
4. ¿Qué sucede con las entradas de control de modo y con la pelota si se oprime el pulsador cuando la pelota se está moviendo hacia la izquierda? ¿Y si la pelota se está moviendo hacia la derecha pero todavía no ha llegado al extremo?
5. Suponga que la pelota volvió a la posición de extrema derecha, pero todavía no se ha accionado el pulsador; ¿en qué estado están las entradas de control de modo si se acciona el pulsador? ¿Qué sucede si no se acciona?

Cómo jugar

Conecte el circuito de la figura 11-20. Pruebe que su funcionamiento sea correcto jugando el juego. El pulsador debe generar una transición de borde positivo y tanto el interruptor de restablecimiento como el de inicio deben estar abiertos (en el estado 1 lógico) durante el juego. Comience con una frecuencia de reloj baja y auméntela para hacer más difícil el juego.

Conteo del número de derrotas

Diseñe un circuito que lleve la cuenta del número de veces que el jugador pierde en el juego. Utilice un decodificador de BCD a siete segmentos y un display de siete segmentos como en la figura 11-8 para mostrar la cuenta de 0 a 9. El conteo se efectúa con un contador decimal utilizando el 7493 como contador decimal de rizo o el 74161 y una compuerta NAND como contador decimal sincrónico. El display deberá indicar 0 cuando el circuito se restablece. Cada vez que la "pelota" desaparece y la luz se apaga, el display deberá incrementarse en 1. Si la luz se mantiene encendida durante el juego, el número del display no deberá cambiar. El diseño final debe ser un circuito de puntaje automático, en el que el display decimal se incrementa automáticamente cada vez que el jugador pierde cuando la luz desaparece.

Tenis de mesa con lámparas

Modifique el circuito de la figura 11-20 para tener un juego de tenis de mesa. Dos jugadores pueden participar en este juego, cada uno con su propio pulsador. El jugador que usa el pulsador derecho devuelve la pelota cuando está en la posición de extrema derecha, y el que usa el pulsador izquierdo lo hace cuando la pelota está en la extrema izquierda. La única modificación requerida es un segundo pulsador y cambiar unos cuantos alambres.

Con un segundo circuito de inicio, cualquiera de los dos jugadores podrá “servir”. Esta adición es opcional.

11-15 GENERADOR DE PULSOS DE RELOJ

En este experimento se utilizará un CI temporizador, conectándolo de modo que genere pulsos de reloj de una frecuencia dada. El circuito requiere dos resistores externos y dos condensadores externos. Se usa el osciloscopio de rayos catódicos para observar las formas de onda y medir la frecuencia.

CI temporizador

El CI tipo 72555 (o 555) es un circuito temporizador de precisión cuya lógica interna se representa en la figura 11-21. (Los resistores R_A y R_B y los dos condensadores no forman parte del CI.) Consiste en dos comparadores de voltaje, un flip-flop y un transistor interno. La división de voltaje de $V_{CC} = 5$ V a través de los tres resistores internos a tierra produce $\frac{2}{3}$ y $\frac{1}{3}$ de V_{CC} (3.3 V y 1.7 V) en las entradas fijas de los comparadores. Cuando la entrada de umbral en la terminal 6 rebasa 3.3 V, el comparador superior restablece el flip-flop y la salida baja a aproximadamente 0 V. Cuando la entrada de disparo en la terminal 2 baja de 1.7 V, el comparador inferior establece el flip-flop y la salida sube a aproximadamente 5 V. Cuando la salida es baja, Q' es alta y la unión base-emisor del transistor está polarizada en directo. Cuando la salida es alta, Q' es baja y el transistor está en corte (véase la sección 10-2). El circuito temporizador puede generar retardos exactos controlados por un circuito *RC* externo. En este experimento, el CI temporizador se operará en el modo astable para generar pulsos de reloj.

Operación del circuito

La figura 11-21 muestra las conexiones externas para el funcionamiento astable. El condensador C se carga a través de los resistores R_A y R_B cuando el transistor está en corte y se descarga a través de R_B cuando el transistor está polarizado en directo y conduce. Cuando el voltaje de carga en el condensador C alcanza los 3.3 V, la entrada de umbral en la terminal 6 hace que el flip-flop se restablezca y el transistor se encienda. Cuando el voltaje de descarga llega a 1.7 V, la entrada de disparo en la terminal 2 hace que el flip-flop se establezca y el transistor se apague. Por tanto, la salida alterna continuamente entre dos niveles de voltaje en la salida del flip-flop. La salida se mantiene alta durante un tiempo igual al tiempo de cargado. Esta duración se determina por la ecuación

$$t_H = 0.693(R_A + R_B)C$$

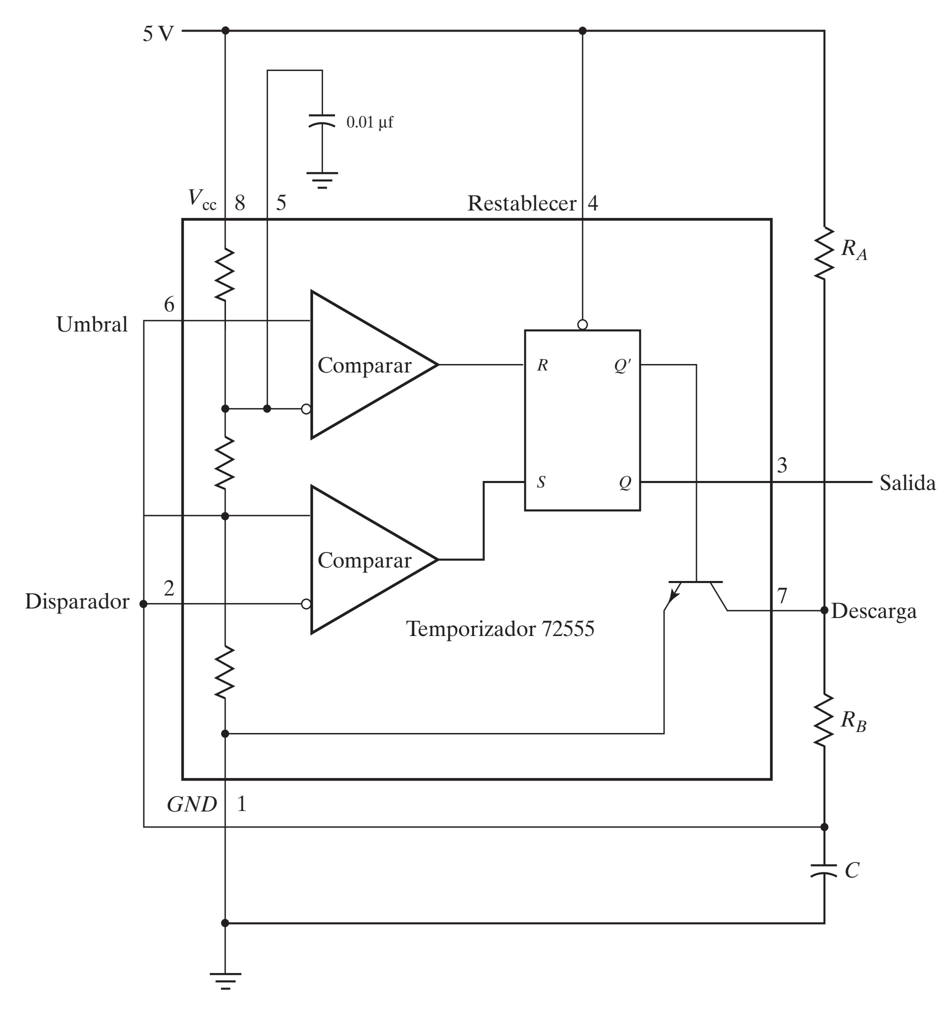


FIGURA 11-21
CI tipo 72555: temporizador conectado como generador de pulsos de reloj

La salida se mantiene baja durante un tiempo igual al tiempo de descarga. Esta duración se determina por la ecuación

$$t_L = 0.693 R_B C$$

Generador de pulsos de reloj

Comenzando con un condensador C de 0.001 μF, calcule valores para R_A y R_B que produzcan pulsos de reloj, como se indica en la figura 11-22. La anchura de pulso es de 1 μs en el nivel bajo, y se repite con una frecuencia de 100 kHz (cada 10 μs). Conecte el circuito y verifique la salida en el osciloscopio.

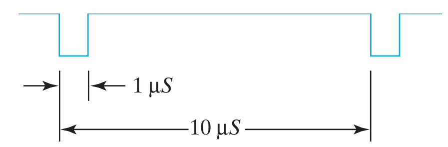


FIGURA 11-22
Forma de onda de salida del generador de reloj

Observe la salida en el condensador *C* y registre sus dos niveles para verificar que estén entre los valores de disparador y de umbral.

Observe la forma de onda en el colector del transistor (terminal 7) y registre toda la información pertinente. Explique la forma de onda analizando la acción del circuito.

Conecte un resistor variable (potenciómetro) en serie con R_A para producir un generador de pulsos de frecuencia variable. La duración del nivel bajo seguirá siendo 1 μs. La frecuencia deberá variar entre 20 y 100 kHz.

Cambie los pulsos de nivel bajo a pulsos de nivel alto con un inversor 7404. Esto producirá pulsos positivos de 1 μs con un intervalo de frecuencia variable.

11-16 SUMADOR PARALELO Y ACUMULADOR

En este experimento construiremos un sumador paralelo de cuatro bits cuya suma se puede cargar en un registro. Los números a sumar se almacenarán en una memoria de acceso aleatorio. Se escogerá un par de números binarios de la memoria y su suma se acumulará en el registro.

Diagrama de bloques

Utilice el circuito de RAM del experimento con memoria de la sección 11-13, un sumador paralelo de 4 bits, un registro de desplazamiento de 4 bits con carga paralela, un flip-flop de acarreo y un multiplexor, para construir el circuito. El diagrama de bloques y los CI a utilizar se ilustran en la figura 11-23. Se puede escribir en la RAM información alimentada con cuatro interruptores o datos de 4 bits que están disponibles en las salidas del registro. La selección se efectúa con un multiplexor. Los datos en RAM se suman al contenido del registro, y la suma se transfiere de vuelta al registro.

Control del registro

Incluya interruptores de dos posiciones para controlar el registro 74194 y el flip-flop de acarreo 7476 así:

a) Una condición de CARGAR para transferir la suma al registro y el acarreo de salida al flip-flop cuando se aplica un pulso de reloj.

b) Una condición DESPLAZAR para desplazar el registro a la derecha, transfiriendo el bit del flip-flop de acarreo a la posición de extrema izquierda del registro, cuando se aplica un pulso de reloj. El valor en el flip-flop de acarreo no deberá cambiar durante el desplazamiento.

c) Una condición SIN CAMBIO que deja el contenido del registro y del flip-flop como estaban, aunque se apliquen pulsos de reloj.

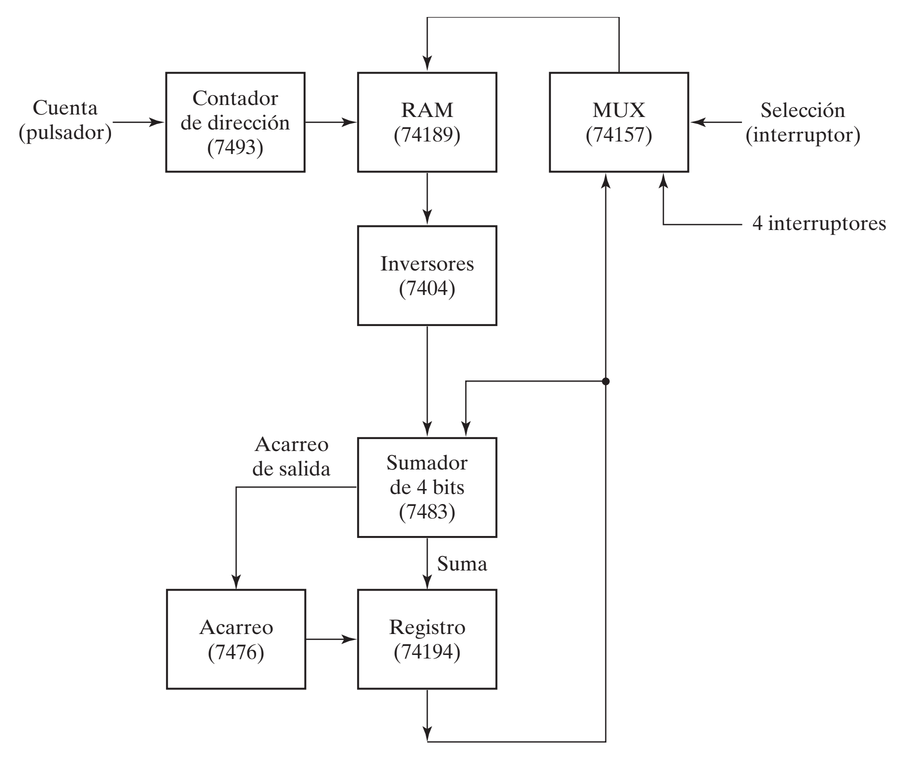


FIGURA 11-23
Diagrama de bloques del sumador paralelo para el experimento 16

Circuito de acarreo

Para ajustarse a las especificaciones anteriores, es necesario incluir un circuito entre el acarreo de salida del sumador y las entradas J y K del flip-flop 7476 para que el acarreo de salida se transfiera al flip-flop (sea 0 o 1) sólo cuando se active la condición CARGAR y se aplique un pulso a la entrada de reloj del flip-flop. El flip-flop de acarreo no deberá cambiar si la condición CARGAR se inhabilita o si se habilita la condición DESPLAZAR.

Circuito detallado

Dibuje un diagrama detallado que muestre todas las conexiones entre los CI. Conecte el circuito e incluya lámparas indicadoras para las salidas del registro y del flip-flop de acarreo, y para la dirección y los datos de salida de la RAM.

Verificación del circuito

Almacene los números siguientes en la RAM y luego súmelos al registro uno por uno. Comience con el registro y el flip-flop despejados. Prediga los valores en la salida del registro y el acarreo después de cada suma, y verifique sus resultados:

$$0110 + 1110 + 1101 + 0101 + 0011$$

Funcionamiento del circuito

Ponga en ceros el registro y el flip-flop de acarreo y almacene los números de 4 bits siguientes en la RAM en las direcciones que se indican:

Dirección	Contenido
0	0110
3	1110
6	1101
9	0101
12	0011

Ahora efectúe estas cuatro operaciones:

1. Sume el contenido de la dirección 0 al contenido del registro utilizando la condición CARGAR.
2. Almacene la suma que está en el registro en la dirección 1 de la RAM.
3. Desplace a la derecha el contenido del registro y el acarreo, con la condición DESPLAZAR.
4. Almacene el contenido desplazado del registro en la dirección 2 de la RAM.

Verifique que el contenido de las tres primeras direcciones de la RAM sea:

Dirección	Contenido
0	0110
1	0110
2	0011

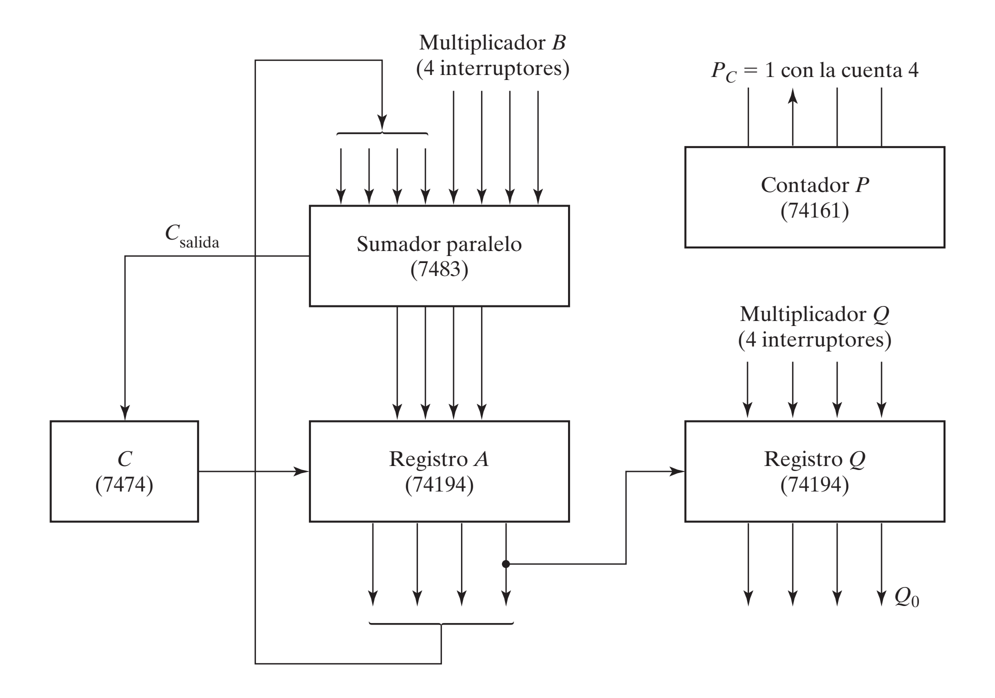
Repita las cuatro operaciones anteriores para cada uno de los otros cuatro números binarios almacenados en RAM. Utilice las direcciones 4, 7, 10 y 13 para almacenar la suma del registro en el paso 2. Use las direcciones 5, 8, 11 y 14 para almacenar el valor desplazado del registro en el paso 4. Prediga el contenido de las direcciones de RAM 0 a 14 y verifique sus resultados.

11-17 MULTIPLICADOR BINARIO

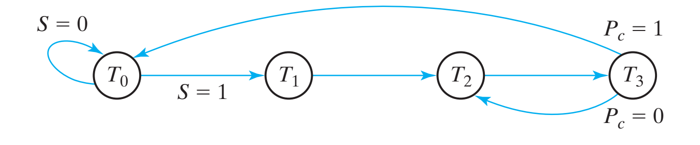
En este experimento se diseñará y construirá un circuito que multiplica dos números de 4 bits sin signo para dar un producto de 8 bits. En la sección 8-6 se presenta un algoritmo para multiplicar dos números binarios.

Diagrama de bloques

En la figura 11-24a) se observa el diagrama de bloques del multiplicador binario con los CI que se recomienda usar. El multiplicando B se obtiene de cuatro interruptores, no de un registro. El multiplicador Q se obtiene de otro juego de cuatro interruptores. El producto se exhibe con ocho lámparas indicadoras. El contador P se inicia en 0 y luego se incrementa después de formarse cada producto parcial. Cuando el contador llega a 4, la salida P_c se vuelve 1 y la operación de multiplicación termina.



a) Diagrama de bloques de la trayectoria de datos



T_1: $A \leftarrow 0, C \leftarrow 0, P \leftarrow 0, Q \leftarrow$ Multiplicador

T_2: $P \leftarrow P + 1$

T_2Q_0: $A \leftarrow A + B, C \leftarrow C_{\text{salida}}$

T_3: shr CAQ, $\text{C} \leftarrow 0$

b) Diagrama de estados de control

FIGURA 11-24
Circuito multiplicador binario

Control de los registros

El diagrama ASM para el multiplicador binario, figura 8-14, muestra que los tres registros y el flip-flop de acarreo se controlan con las señales T_1, T_2 y T_3. Una señal de control adicional que depende de Q_1 carga la suma en el registro A y el acarreo de salida en el flip-flop C. Q_0 es el bit menos significativo del registro Q. El diagrama de estados de control y las operaciones a efectuar en cada estado se dan en la figura 11-24b). T_2Q_0 se genera con una compuerta AND

cuyas entradas son T_2 y Q_0. El flip-flop de acarreo C se puede poner en 0 con cada pulso de reloj, excepto cuando el acarreo de salida se transfiere a él.

Ejemplo de multiplicación

Antes de conectar el circuito, asegúrese de comprender el funcionamiento del multiplicador. Para ello, construya una tabla similar a la de la tabla 8-4, pero con $B = 1111$ como multiplicando y $Q = 1011$ como multiplicador. Junto a cada uno de los comentarios de la columna izquierda de la tabla, especifique cuál de las variables de estado —T_1 o T_2 o T_3— está habilitada en cada caso. (Los estados deberán iniciar con T_1 y luego repetir T_2 y T_3 cuatro veces.)

Diseño de la trayectoria de datos

Dibuje un diagrama detallado de la parte de la trayectoria de datos del multiplicador, mostrando todas las conexiones a terminales del CI. Genere las señales de control —T_1, T_2 y T_3— con tres interruptores y utilícelas para aplicar las operaciones de control requeridas a los diversos registros. Conecte el circuito y compruebe que cada componente esté funcionando como es debido. Con las tres variables de control en 0, ponga los interruptores del multiplicando en 1111, y los del multiplicador, en 1011. Genere manualmente la sucesión de variables de control con los interruptores, como especifica el diagrama de estados de la figura 11-24b). Aplique un solo pulso estando en cada estado de control y observe las salidas de los registros A y Q, y los valores de C y P_c. Compárelos con los valores del ejemplo numérico para verificar que el circuito esté funcionando correctamente. Cabe señalar que el CI tipo 74161 tiene flip-flops amo-esclavo. Para operarlo manualmente, es necesario que el pulso de reloj individual sea negativo.

Diseño del control

Diseñe el circuito de control especificado por el diagrama de estados. Utilice cualquiera de los métodos de implementación de control que se describen en la sección 8-7.

Escoja el método que reduce al mínimo el número de circuitos integrados. Verifique el funcionamiento del circuito de control antes de conectarlo al procesador de datos.

Verificación del multiplicador

Conecte las salidas del circuito de control al procesador de datos y verifique el funcionamiento del circuito total repitiendo los pasos de la multiplicación de 1111 por 1011. Ahora, los pulsos de reloj individuales también deberán hacer que se genere la sucesión de estados de control (elimine los interruptores manuales). La señal de inicio S se genera con un interruptor que está cerrado mientras el control está en el estado T_0.

Genere la señal de inicio S con un pulsador o cualquier otro pulso corto y opere el multiplicador con pulsos de reloj continuos producidos por un generador de reloj. Un accionamiento del pulsador S deberá iniciar la operación de multiplicación; al final, se deberá exhibir el producto en los registros A y Q. La multiplicación se repetirá en tanto esté habilitada la señal S. Asegúrese de que S vuelva a 0, ajuste los interruptores de modo que produzcan otros dos números de cuatro bits, y oprima S otra vez. El nuevo producto deberá aparecer en las salidas. Multiplique otros pares de números hasta convencerse de que el circuito funciona correctamente.

11-18 CIRCUITOS SECUENCIALES ASINCRÓNICOS

En este experimento se analizarán y diseñarán circuitos secuenciales asincrónicos. Estos tipos de circuitos se presentan en el capítulo 9.

Ejemplo de análisis

El análisis de los circuitos secuenciales asincrónicos con latches *SR* se bosqueja en la sección 9-3. Analice el circuito de la figura P9-9 (que acompaña al problema 9-9) deduciendo la tabla de transición y el mapa de salida del circuito. A partir de esa tabla y mapa, determine: a) ¿qué sucede con la salida *Q* cuando la entrada x_1 es 1, independientemente del valor de la entrada x_2?; b) ¿qué sucede con la salida *Q* cuando la entrada x_2 es 1 y x_1 es 0?, y c) ¿qué sucede con la salida *Q* cuando ambas entradas vuelven a 0?

Conecte el circuito y demuestre que opera según el análisis.

Ejemplo de diseño

En la figura 5-10 se ilustra el circuito de un flip-flop tipo *D* disparado por borde positivo. En la figura 9-46 se aprecia el circuito de un flip-flop tipo *T* disparado por borde negativo. Utilizando el procedimiento de seis pasos recomendado en la sección 9-8, diseñe, construya y pruebe un flip-flop tipo *D* que se dispare con ambas transiciones del reloj, positiva y negativa. El circuito tiene dos entradas —D y *C*— y una sola salida, *Q*. El valor de *D* en el momento en que *C* cambia de 0 a 1 se convierte en la salida del flip-flop, *Q*. La salida no cambiará, sea cual sea el valor de *D*, en tanto *C* permanezca en 1. En la siguiente transición de reloj, la salida se volverá a actualizar con el valor de *D* cuando *C* cambie de 1 a 0. Después, la salida se mantendrá sin cambio en tanto *C* siga siendo 0.

11-19 EXPERIMENTOS DE SIMULACIÓN EN VERILOG HDL

Algunos de los experimentos con hardware delineados en este capítulo se pueden complementar con un procedimiento correspondiente en software empleando el Lenguaje de Descripción de Hardware (HDL) Verilog. Se requiere un compilador de Verilog y un simulador para este complemento. He aquí algunas sugerencias para simular y probar algunos de los circuitos empleados en los experimentos de laboratorio.

Complemento del experimento 2 (sección 11-2)

Las diversas compuertas lógicas y sus retardos de propagación se presentaron en el experimento con hardware. En la sección 3-9 se investigó un sencillo circuito con retardos de compuerta. Como introducción al programa Verilog de laboratorio, compile el circuito descrito en el ejemplo HDL 3-3 y luego ejecute el simulador para verificar las formas de onda que se muestran en la figura 3-38.

Asigne los retardos siguientes al circuito OR exclusivo de la figura 3-32a): 10 ns para los inversores, 20 ns para las compuertas AND y 30 ns para las compuertas OR. La entrada del circuito cambia de $xy = 00$ a $xy = 01$.

- **a)** Determine las señales que hay en la salida de cada compuerta entre $t = 0$ y $t = 50$ ns.
- **b)** Escriba la descripción HDL del circuito, incluidos los retardos.
- **c)** Escriba un módulo de estímulo (similar al ejemplo HDL 3-3) y simule el circuito para verificar la respuesta a la parte a).

Complemento del experimento 4 (sección 11-4)

El funcionamiento de un circuito combinacional se verifica examinando la salida y comparándola con la tabla de verdad del circuito. El ejemplo HDL 4-10 (sección 4-11) ilustra el procedimiento para obtener por simulación la tabla de verdad de un circuito combinacional. Para familiarizarse con este procedimiento, compile y simule el ejemplo HDL 4-10 y compruebe la tabla de verdad de salida.

En el experimento 4 se diseñó un circuito lógico mayoritario. Escriba la descripción HDL en el nivel de compuertas de ese circuito, junto con un estímulo para exhibir la tabla de verdad. Compile y simule el circuito y verifique la salida.

Complemento del experimento 5 (sección 11-5)

Este experimento se ocupa de la conversión de códigos. En la sección 4-3 se diseñó un convertidor de BCD a exceso-3. Utilice el resultado del diseño para verificarlo con un simulador HDL.

a) Escriba una descripción HDL en el nivel de compuertas del circuito que se muestra en la figura 4-4.

b) Escriba una descripción de flujo de datos utilizando las expresiones booleanas de la figura 4-3.

c) Escriba una descripción HDL del comportamiento de un convertidor BCD a exceso-3.

d) Escriba un conjunto de pruebas para simular y probar el circuito convertidor de BCD a exceso-3 y verificar la tabla de verdad. Verifique los tres circuitos.

Complemento del experimento 7 (sección 11-7)

En este experimento se crea un sumador-restador de 4 bits. También se crea un circuito sumador-restador en la sección 4-4.

a) Escriba la descripción HDL del comportamiento del sumador de 4 bits 7483.

b) Escriba una descripción del comportamiento del circuito sumador-restador que se ilustra en la figura 11-11.

c) Escriba la descripción HDL jérárquica del sumador-restador de 4 bits que se representa en la figura 4-13 (incluida *V*). Esto puede hacerse creando un ejemplar de una versión modificada del sumador de 4 bits descrito en el ejemplo HDL 4-2 (sección 4-11).

d) Escriba un conjunto de pruebas HDL para simular y probar los circuitos de la parte c). Verifique los valores que causan un desbordamiento con $V = 1$.

Complemento del experimento 8 (sección 11-8)

El flip-flop *D* disparado por flanco 7474 se muestra en la figura 11-13. El flip-flop tiene entradas de preestablecimiento y despeje asincrónicos.

a) Escriba una descripción HDL del comportamiento del flip-flop *D* 7474 utilizando únicamente la salida *Q*. (Observe que, cuando Preestablecer = 0, *Q* cambia a 1, y cuando Preestablecer = 1 y Despejar = 0, *Q* cambia a 0. Es decir, Preestablecer tiene precedencia sobre Despeje.)

b) Escriba una descripción HDL del comportamiento del flip-flop *D* 7474 empleando ambas salidas. Rotule con *Q_not* la segunda salida, teniendo presente que no siempre es el complemento de *Q*. (Cuando Preestablecer = Despejar = 0, tanto *Q* como *Q_not* cambian a 1.)

Complemento del experimento 9 (sección 11-9)

En el experimento con hardware se pide diseñar y probar un circuito secuencial cuyo diagrama de estados aparece en la figura 11-14. Se trata de un circuito secuencial de modelo Mealy similar al que se describe en el ejemplo HDL 5-5 (sección 5-5).

a) Escriba la descripción HDL del diagrama de estados de la figura 11-14.

b) Escriba la descripción HDL estructural del circuito secuencial obtenido del diseño. (Es similar al ejemplo HDL 5-7, sección 5-5.)

c) La figura 11-24b) (sección 11-17) muestra un diagrama de estados de control. Escriba la descripción HDL del diagrama de estados utilizando la asignación binaria de un solo uno (véase la tabla 5-9 en la sección 5-6) y tres salidas, T_1, T_2 y T_3.

Complemento del experimento 10 (sección 11-10)

El CI tipo 74161, un contador sincrónico con carga paralela, se representa en la figura 11-15. Es similar al descrito en el ejemplo HDL 6-3 (sección 6-6) con dos excepciones. La entrada de carga se habilita cuando es 0, y hay dos entradas (P y T) que controlan el conteo. Escriba la descripción HDL del CI 74161.

Complemento del experimento 11 (sección 11-11)

En este experimento se diseña un registro de desplazamiento bidireccional con carga paralela, utilizando los CI tipo 74195 y 74157.

a) Escriba la descripción HDL del registro de desplazamiento 74195. Suponga que las entradas J y $\overline{K}$ se conectan entre sí para formar la entrada en serie.

b) Escriba la descripción HDL del multiplexor 74157.

c) Escriba la descripción HDL del registro de desplazamiento bidireccional de 4 bits diseñado en este experimento. 1) Escriba la descripción estructural creando un ejemplar de los dos CI y especificando la interconexión, y 2) escriba la descripción del comportamiento del circuito utilizando la tabla de función deducida en el experimento de diseño.

Complemento del experimento 13 (sección 11-13)

Este experimento investiga el funcionamiento de una memoria de acceso aleatorio (RAM). La forma de describir una memoria en HDL se explica en la sección 7-2 y se ilustra con el ejemplo HDL 7-1.

a) Escriba la descripción HDL del CI de RAM tipo 74189 que se muestra en la figura 11-18.

b) Pruebe el funcionamiento de la memoria escribiendo un programa de estímulo que almacene un 3 binario en la dirección 0 y un 1 binario en la dirección 14. Luego lea el contenido de esas dos direcciones para comprobar que los números se hayan almacenado correctamente.

Complemento del experimento 14 (sección 11-14)

Escriba la descripción HDL del comportamiento del registro de desplazamiento bidireccional con carga paralela 74194 que se presenta en la figura 11-19.

Complemento del experimento 16 (sección 11-16)

En el diagrama de bloques de la figura 11-23 se reproduce un sumador paralelo con registro acumulador y unidad de memoria. Escriba la descripción estructural del circuito especificado por el diagrama de bloques. Dicha descripción HDL se obtiene creando ejemplares de los diversos componentes. El ejemplo HDL 4-8 de la sección 8-5 ilustra la descripción estructural de un diseño. Primero hay que escribir la descripción del comportamiento de cada componente. Use el contador 74161 en vez del 7493, y sustituya el flip-flop *JK* 7476 por un flip-flop *D* 7474. La lista de la tabla 11-1 indica dónde están los diagramas de bloques de los diversos componentes.

Complemento del experimento 17 (sección 11-17)

En la figura 11-24 se aprecia el diagrama de bloques de un multiplicador binario de 4 bits. Hay dos formas de describir el multiplicador: 1) utilizando los enunciados de nivel de transferencia de registros que se dan en la parte b) de la figura o 2) utilizando el diagrama de bloques de la parte a) de la figura. La descripción del multiplicador en términos del formato de nivel de transferencia de registros (RTL) se efectúa en el ejemplo HDL 8-5 (sección 8-7). En este experimento, usaremos los componentes de circuito integrado especificados en el diagrama de bloques para escribir la descripción HDL estructural del multiplicador binario. Dicha descripción se obtiene a partir del módulo que describe cada componente, creando ejemplares de cada uno para mostrar su interconexión. (Se da un ejemplo en la sección 8-5.) Las descripciones HDL de los componentes podrían obtenerse de las soluciones a experimentos anteriores. La descripción del 7483 se pide en el experimento 7a), la del 7474, en el experimento 8a), la del 74161 en el experimento 10, la del 74194 en el experimento 14, y la del control, en el experimento 9c).

12 Símbolos gráficos estándar

12-1 SÍMBOLOS RECTANGULARES

Los componentes digitales, como compuertas, decodificadores, multiplexores y registros, se pueden obtener comercialmente como circuitos integrados y se clasifican como circuitos SSI o MSI. Se han creado símbolos gráficos estándar para estos y otros componentes, que le permitan al usuario reconocer cada función a partir del símbolo gráfico singular que se le asigna. Esta norma, llamada ANSI/IEEE Std. 91-1984, ha sido aprobada por la industria, el gobierno y organizaciones profesionales, y es congruente con normas internacionales.

La norma utiliza un contorno rectangular para representar cada función lógica. Dentro del contorno, hay un símbolo calificador general que denota la operación lógica que la unidad efectúa. Por ejemplo, el símbolo calificador general para un multiplexor es MUX. El tamaño del contorno es arbitrario y puede ser cuadrado o rectangular, con una razón longitud/anchura arbitraria. Las líneas de entrada se colocan a la izquierda, y las de salida, a la derecha. Si la dirección de flujo de la señal se invierte, se deberá indicar con flechas.

Los símbolos gráficos rectangulares para las compuertas SSI se representan en la figura 12-1. El símbolo calificador para la compuerta AND es &. La compuerta OR tiene el símbolo calificador que denota mayor o igual que 1, lo que indica que por lo menos una entrada debe estar activa para que la salida esté activa. El símbolo para la compuerta búfer es 1, e indica que sólo hay una entrada. El símbolo de OR exclusivo indica el hecho de que sólo puede estar activa una entrada para que la salida esté activa. La inclusión de la burbuja de negación lógica en la salida convierte a las compuertas en sus complementos. Aunque se recomienda usar los símbolos rectangulares para las compuertas, la norma también reconoce los símbolos de forma distintiva para las compuertas que se presentan en la figura 2-5.

Un ejemplo de símbolo gráfico estándar para MSI es el sumador paralelo de cuatro bits que se aprecia en la figura 12-2. El símbolo calificador para un sumador es la letra griega Σ. Las letras preferidas para los operandos aritméticos son P y Q. Los símbolos que agrupan bits en los dos tipos de entrada y en la salida de suma son los equivalentes decimales de los pesos de

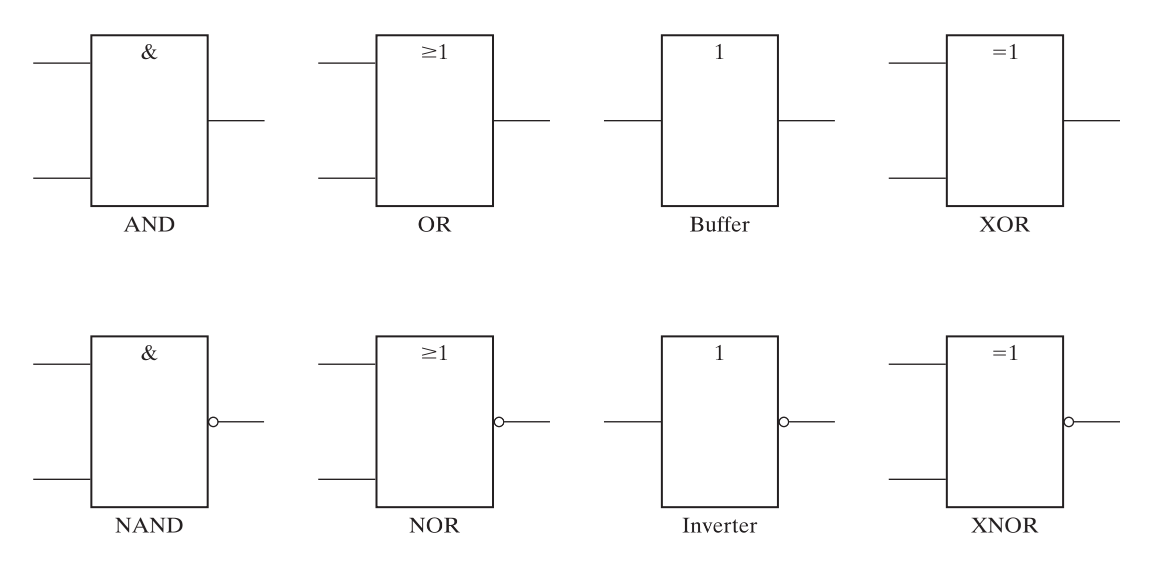


FIGURA 12-1
Símbolos gráficos rectangulares para las compuertas

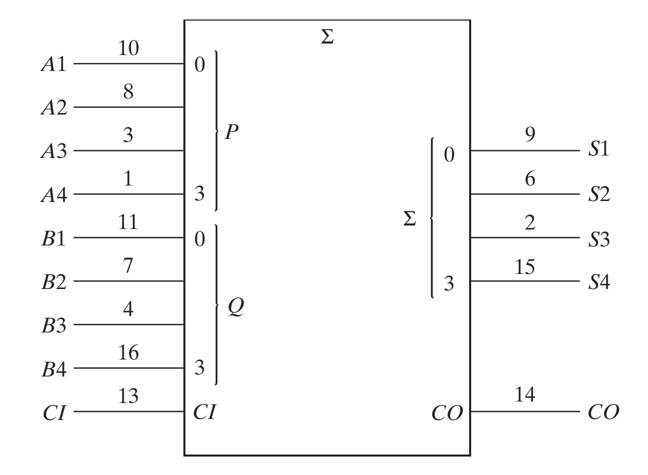


FIGURA 12-2
Símbolo gráfico estándar para un sumador paralelo de 4 bits, CI tipo 7483

los bits como exponentes de 2. Así pues, la entrada rotulada 3 corresponde al valor de $2^3 = 8$. El acarreo de entrada se designa con *CI*, y el de salida, con *CO*. Si el componente digital representado por el contorno también es un circuito integrado comercial, se acostumbra escribir el número de terminal junto a cada entrada y salida. Así, el CI tipo 7483 es un sumador de 4 bits con acarreo anticipado. Está encerrado en un paquete de 16 terminales. Los números de terminal para las nueve entradas y las cinco salidas se muestran en la figura 12-2. Las otras dos terminales son para la fuente de poder.

Antes de presentar los símbolos gráficos de otros componentes, es necesario repasar algo de terminología. Como se mencionó en la sección 2-7, un sistema de lógica positiva define el más positivo de dos niveles de señal (denotado por H) como 1 lógico, y el más negativo (designado por L), como 0 lógico. La lógica negativa supone la asignación opuesta. Una tercera alternativa es utilizar una convención de lógica mixta, donde las señales se consideran exclusivamente en términos de sus valores H y L. En cualquier punto del circuito, el usuario puede definir la polaridad lógica asignando 1 lógico a la señal H o a la L. La notación de lógica mixta utiliza un pequeño símbolo gráfico de triángulo rectángulo para indicar polaridad de lógica negativa en cualquier terminal de entrada o de salida. [Véase la figura 2-10f).]

Los fabricantes de circuitos integrados especifican el funcionamiento de los circuitos en términos de señales H y L. Cuando una entrada o salida se considera en términos de lógica positiva, se define como *activa-alta*. Cuando se considera en términos de lógica negativa, se define como *activa-baja*. Las entradas o salidas activas-bajas se reconocen por la presencia del pequeño símbolo triangular indicador de polaridad. Cuando se usa lógica positiva exclusivamente en todo el sistema, el símbolo triangular de polaridad equivale al pequeño círculo (burbuja) que denota negación. En este libro hemos supuesto lógica positiva siempre y hemos utilizado la burbuja al dibujar diagramas lógicos. Cuando una línea de entrada o salida no incluye la burbuja, la definimos como activa si es 1 lógico. Una entrada o salida que lleva burbuja se considera activo si está en el estado 0 lógico. Sin embargo, utilizaremos el símbolo de polaridad triangular para indicar asignación activa-baja en todos los dibujos que representen diagramas estándar. Esto coincidirá con los libros de datos de circuitos integrados, donde suele utilizarse el símbolo de polaridad. Podríamos haber dibujado las cuatro compuertas de abajo de la figura 12-1 con un pequeño triángulo en las líneas de salida, en lugar de la burbuja.

En la figura 12-3 se presenta otro ejemplo de símbolo gráfico para circuito MSI. Se trata de un decodificador de 2 a 4 líneas que representa una mitad del CI tipo 74155. Las entradas están a la izquierda, y las salidas, a la derecha. El símbolo identificador *X/Y* indica que el circuito convierte del código X al código Y. Se asignan pesos binarios de 1 y 2 a las entradas de datos A y B, equivalentes a 2^0 y 2^1, respectivamente. Se asignan a las salidas números de 0 a 3, que corresponden a las salidas D_0 a D_3, respectivamente. El decodificador tiene una entrada activa baja, E_1, y una activa-alta, E_2. Estas dos entradas pasan por una compuerta AND interna para habilitar el decodificador. La salida de la compuerta AND se rotula *EN* (*enable*, habilitar) y se activa cuando E_1 está en el estado de nivel bajo y E_2 está en el estado de nivel alto.

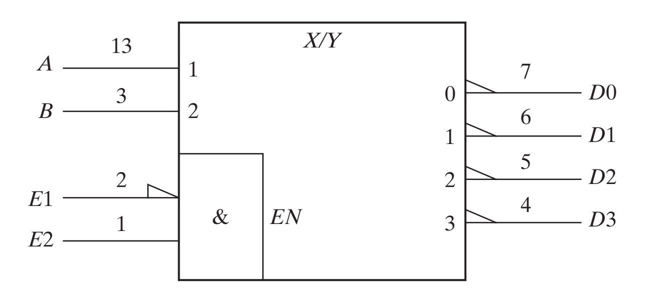


FIGURA 12-3
Símbolo gráfico estándar para un decodificador de 2 a 4 líneas (medio CI tipo 74155)

12-2 SÍMBOLOS CALIFICADORES

Los símbolos gráficos estándar del IEEE para funciones lógicas incluyen una lista de símbolos calificadores que se usan junto con el contorno. Se añade un símbolo calificador al contorno básico para designar las características lógicas generales del elemento o las características físicas de una entrada o salida. En la tabla 12-1 se dan algunos símbolos calificadores generales especificados en la norma. Un símbolo calificador general define la función básica desempeñada por el dispositivo representado en el diagrama. Se coloca cerca de la posición central superior del contorno rectangular. En diagramas anteriores se mostraron los símbolos calificadores generales para las compuertas, decodificadores y sumadores. Los demás símbolos no requieren mayor explicación y se usarán posteriormente en diagramas que representan los elementos digitales correspondientes.

Tabla 12-1
Símbolos calificadores generales

Símbolo	Descripción
&	Compuerta o función AND
≥ 1	Compuerta o función OR
1	Compuerta búfer o inversor
$=1$	Compuerta o función OR exclusivo
2k	Función par o elemento de paridad par
2k + 1	Función impar o elemento de paridad impar
X/Y	Codificador, decodificador o convertidor de código
MUX	Multiplexor
DMUX	Desmultiplexor
Σ	Sumador
Π	Multiplicador
COMP	Comparador de magnitud
ALU	Unidad de aritmética y lógica
SRG	Registro de desplazamiento
CTR	Contador
RCTR	Contador con rizo
ROM	Memoria de sólo lectura
RAM	Memoria de acceso aleatorio

Algunos de los símbolos calificadores asociados a entradas y salidas se incluyen en la figura 12-4. Los símbolos asociados a entradas están en la parte izquierda de la columna rotulada *símbolo*. Los símbolos asociados a salidas están en la parte derecha de la columna. El símbolo de entrada o salida activa-baja es el indicador de polaridad. Como ya se mencionó, equivale a la negación lógica cuando se supone lógica positiva. La entrada dinámica está asociada a la entrada de reloj en los circuitos de flip-flop. Indica que la entrada está activa en una transición de señal de nivel bajo a nivel alto. La salida de tres estados tiene un tercer estado de alta impedancia, que carece de significado lógico. Cuando el circuito está habilitado, la salida está en el estado normal de 0 o 1 lógico, pero cuando está inhabilitado, la salida está en un estado de alta impedancia, que equivale a un circuito abierto.

La salida de colector abierto tiene un estado que exhibe una condición de alta impedancia. En ocasiones se requiere un resistor conectado externamente para generar el nivel lógico apro-

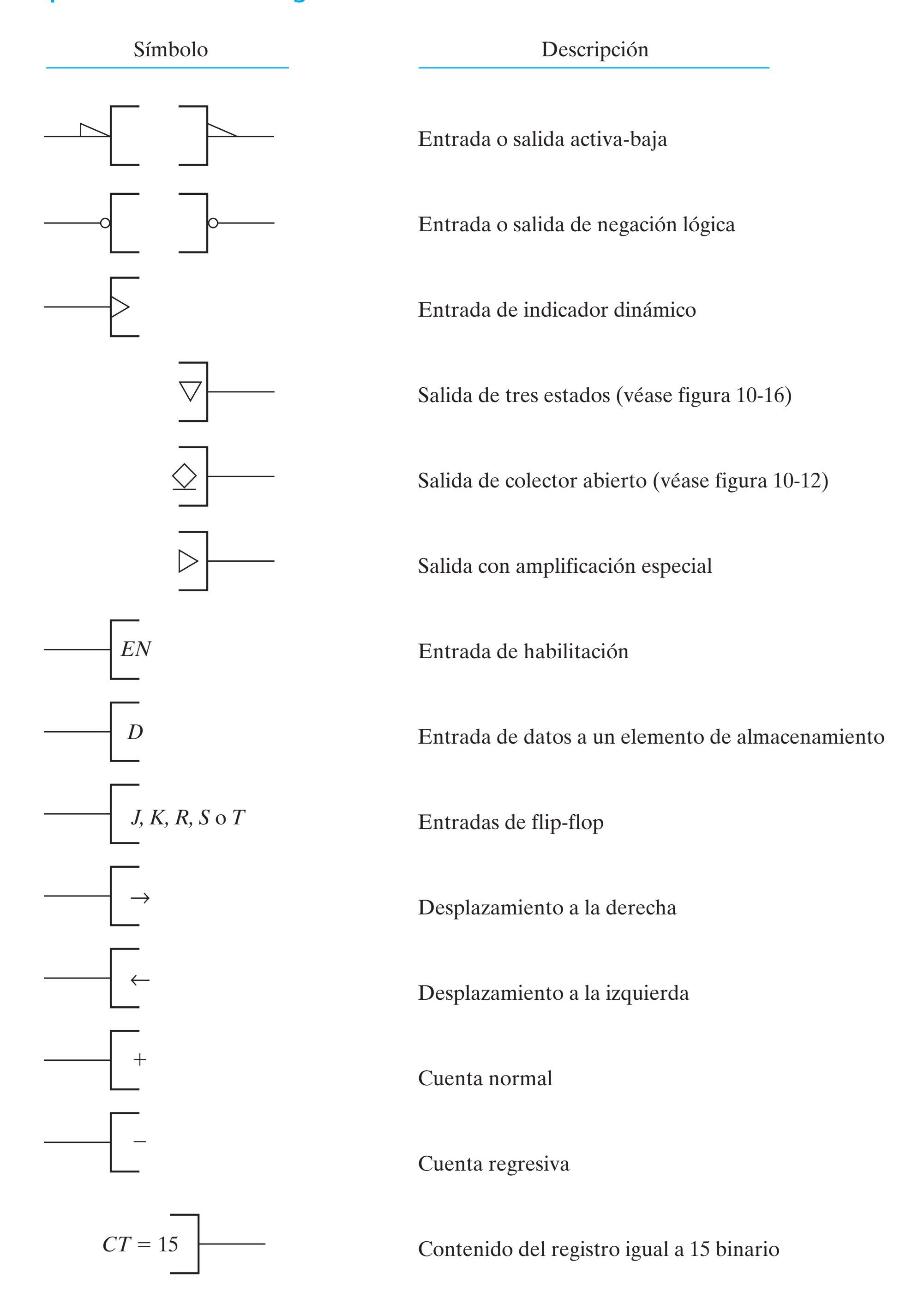


FIGURA 12-4
Símbolos de calificación asociados a entradas y salidas

piado. El símbolo de rombo podría tener una raya arriba (para indicar tipo alto) o abajo (tipo bajo). El tipo alto o bajo especifica el nivel lógico cuando la salida no está en el estado de alta impedancia. Por ejemplo, los circuitos integrados tipo TTL tienen salidas especiales llamadas salidas de colector abierto. Éstas se reconocen por un símbolo de rombo con una raya abajo. Esto indica que la salida puede estar en un estado de alta impedancia o en un estado de nivel bajo. Cuando se usan como parte de una función de distribución, dos o más compuertas NAND de colector abierto conectadas a un resistor común ejecutan una función AND de lógica positiva o una función OR de lógica negativa.

La salida con amplificación especial se usa en compuertas que tienen funciones de alimentación especiales. Tales compuertas se usan en componentes como manejadores de reloj o transmisores orientados a bus. El símbolo *EN* indica una entrada de habilitación. Su efecto es habilitar todas las salidas cuando está activo. Si la entrada marcada con *EN* está inactiva, todas las salidas quedan inhabilitadas. Los símbolos para las entradas de flip-flop tienen el significado acostumbrado. La entrada *D* también está asociada a otros elementos de almacenamiento, como una entrada de memoria.

Los símbolos de desplazamiento a la derecha y a la izquierda son flechas que apuntan a la derecha y a la izquierda, respectivamente. Los símbolos para contadores de cuenta ascendente y descendente son los símbolos más y menos, respectivamente. Una salida designada con $CT = 15$ está activa cuando el contenido del registro llega a la cuenta binaria 15. Si aparece información no estándar dentro del contorno, se le encierra en paréntesis cuadrados, [así].

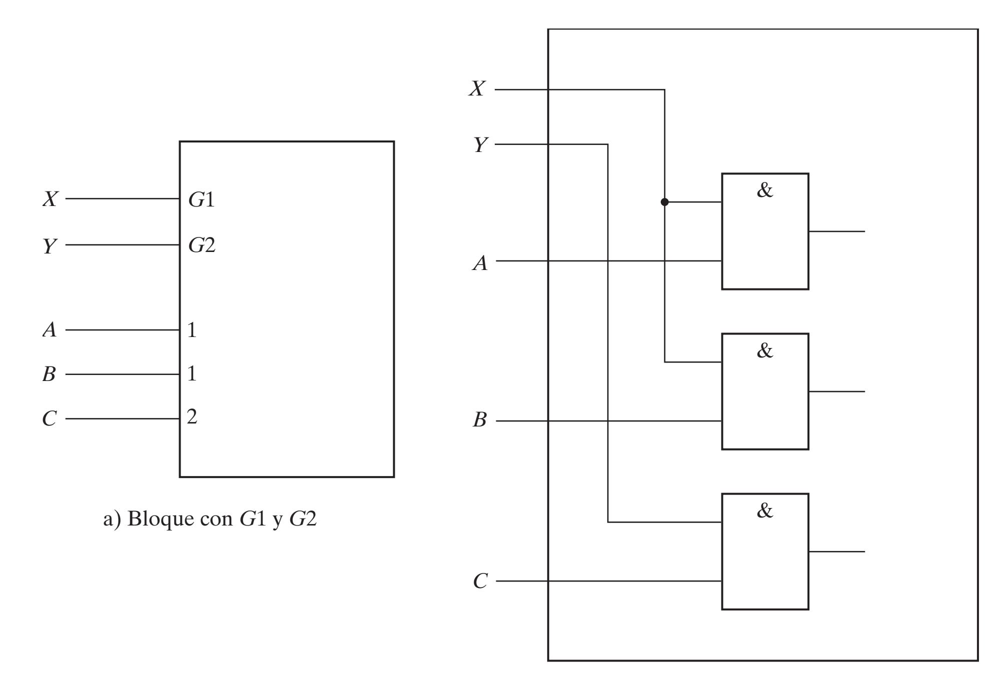
12-3 NOTACIÓN DE DEPENDENCIA

El aspecto más importante de los símbolos lógicos estándar es la notación de dependencia. Esta notación sirve para indicar la relación entre diferentes entradas y salidas sin mostrar realmente todos los elementos e interconexiones. Primero ilustraremos esta notación de dependencia con un ejemplo de la dependencia AND y luego definiremos los demás símbolos asociados con esta notación.

La dependencia AND se representa con la letra *G* seguida de un número. Cualquier entrada o salida de un diagrama que se rotule con el número asociado a *G* se considera en AND con ella. Por ejemplo, si una entrada del diagrama tiene el rótulo *G*1 y otra entrada está rotulada con el número 1, se considera que hay un AND interno de ambas entradas *G*1 y 1.

En la figura 12-5 se presenta un ejemplo de dependencia AND. En a) tenemos una porción de un símbolo gráfico con dos rótulos de dependencia AND, *G*1 y *G*2. Hay dos entradas rotuladas con el número 1 y una rotulada con el número 2. En la parte b) de la figura se muestra la interpretación equivalente. La entrada *X* asociada a *G*1 se considera en AND con las entradas *A* y *B*, rotuladas con 1. Asimismo, la entrada *Y* está en AND con la entrada *C* por la dependencia entre *G*2 y 2.

La norma define otras 10 dependencias. Cada una se denota con un símbolo de letra (excepto *EN*). La letra aparece en la entrada o salida y va seguida de un número. Cada entrada o sa-



a) Bloque con *G*1 y *G*2

b) Interpretación equivalente

FIGURA 12-5
Ejemplo de dependencia *G* (AND)

lida afectada por esa dependencia se rotula con ese mismo número. Las 11 dependencias y su correspondiente designación de letra son:

G	Denota una relación (compuerta) AND
V	Denota una relación OR
N	Denota una relación de negar (OR exclusivo)
EN	Especifica una acción de habilitar
C	Identifica una dependencia de control
S	Especifica una acción de establecer
R	Especifica una acción de restablecer
M	Identifica una dependencia de modo
A	Identifica una dependencia de dirección
Z	Indica una interconexión interna
X	Indica una transmisión controlada

Las dependencias *V* y *N* sirven para denotar las relaciones booleanas OR y OR exclusivo, de forma similar a como *G* denota el AND booleano. La dependencia *EN* es similar al símbolo calificador *EN*, salvo que va seguida de un número (por ejemplo, *EN* 2). Sólo las salidas marcadas con ese número se inhabilitan cuando la entrada asociada a *EN* está activa.

La dependencia de control *C* sirve para identificar una entrada de reloj en un elemento secuencial y para indicar a cuál entrada controla. Las dependencias de establecer (*S*) y restablecer (*R*) sirven para especificar estados lógicos internos de un flip-flop *SR*. Las dependencias *C*, *S* y *R* se explican en la sección 12-5 junto con el circuito de flip-flop. La dependencia de modo, *M*, sirve para identificar entradas que seleccionan el modo de operación de la unidad. La dependencia de modo se presentará en la sección 12-6 en relación con los registros y contadores. La dependencia de dirección *A* sirve para identificar la entrada de dirección de una memoria. La presentaremos en la sección 12-8 junto con la unidad de memoria.

La dependencia *Z* sirve para indicar interconexiones dentro de la unidad. Señala la existencia de conexiones lógicas internas entre entradas, salidas, entradas internas y salidas internas, en cualquier combinación. La dependencia *X* sirve para indicar la trayectoria de transmisión controlada en una compuerta de transmisión CMOS.

12-4 SÍMBOLOS PARA ELEMENTOS COMBINACIONALES

Los ejemplos de esta sección y del resto del capítulo ilustran el uso de la norma para representar diversos componentes digitales con símbolos gráficos. Los ejemplos ilustran circuitos integrados comerciales reales, e incluyen los números de terminal en las entradas y salidas. Casi todos los CI presentados en este capítulo se usan en los experimentos sugeridos en el capítulo 11.

Los símbolos gráficos del sumador y el decodificador se mostraron en la sección 12-2. El CI tipo 74155 se puede conectar como decodificador 3 × 8, como se indica en la figura 12-6. (La tabla de verdad de este decodificador aparece en la figura 11-7.) Hay dos entradas *C* y *G* en el CI. Cada par se debe conectar entre sí como se muestra en el diagrama. La entrada de habilitación está activa cuando está en el estado de bajo nivel. Todas las salidas son activas-bajas. Se asignan a las entradas los pesos binarios 1, 2 y 4, equivalentes a 2^0, 2^1 y 2^2, respectivamente. Se asignan a las salidas números del 0 al 7. La suma de los pesos de las entradas determina qué salida está activa. Por ejemplo, si se activan las líneas de entrada cuyos pesos son 1 y 4, el peso total será $1 + 4 = 5$, y se activará la salida 5. Desde luego, se debe activar la entrada *EN* para que cualquier salida pueda activarse.

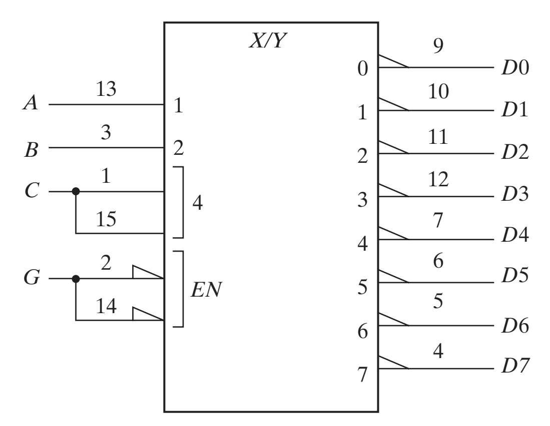


FIGURA 12-6
CI tipo 74155 conectado como decodificador 3 × 8

El decodificador es un caso especial de un componente más general llamado *codificador*. Un codificador es un dispositivo que recibe un código binario por varias entradas y produce un código binario distinto por varias salidas. En lugar de usar el símbolo calificador *X/Y*, el codificador se especifica con el nombre de los códigos. Por ejemplo, el decodificador de 3 a 8 líneas de la figura 12-6 se simboliza con el nombre *BIN/OCT* porque el circuito convierte un número binario de tres bits en 8 valores octales, del 0 al 7.

Antes de mostrar el símbolo gráfico del multiplexor, es necesario mostrar una variación de la dependencia AND. Esta dependencia a veces se representa con una notación abreviada como $G\frac{0}{7}$. Este símbolo representa ocho símbolos de dependencia AND, de 0 a 7, a saber:

$$G0, G1, G2, G3, G4, G5, G6, G7$$

En cualquier momento dado, sólo una de las ocho compuertas AND puede estar activa. La compuerta AND activa se determina a partir de las entradas asociadas al símbolo *G*. Estas entradas se marcan con pesos iguales a las potencias de 2. Para las ocho compuertas AND que acabamos de numerar, los pesos son 0, 1 y 2, que corresponden a los números 2^0, 2^1 y 2^2, respectivamente. La compuerta AND que está activa en un momento dado se determina a partir de la suma de los pesos de las entradas activas. Por ejemplo, si están activas las entradas 0 y 2, la compuerta AND activa tiene el número $2^0 + 2^2 = 5$. Esto activa a *G*5 y desactiva a las otras siete compuertas AND.

El símbolo gráfico estándar de un multiplexor 8 × 1 se muestra en la figura 12-7a). El símbolo calificador MUX identifica el dispositivo como un multiplexor. Los símbolos dentro del

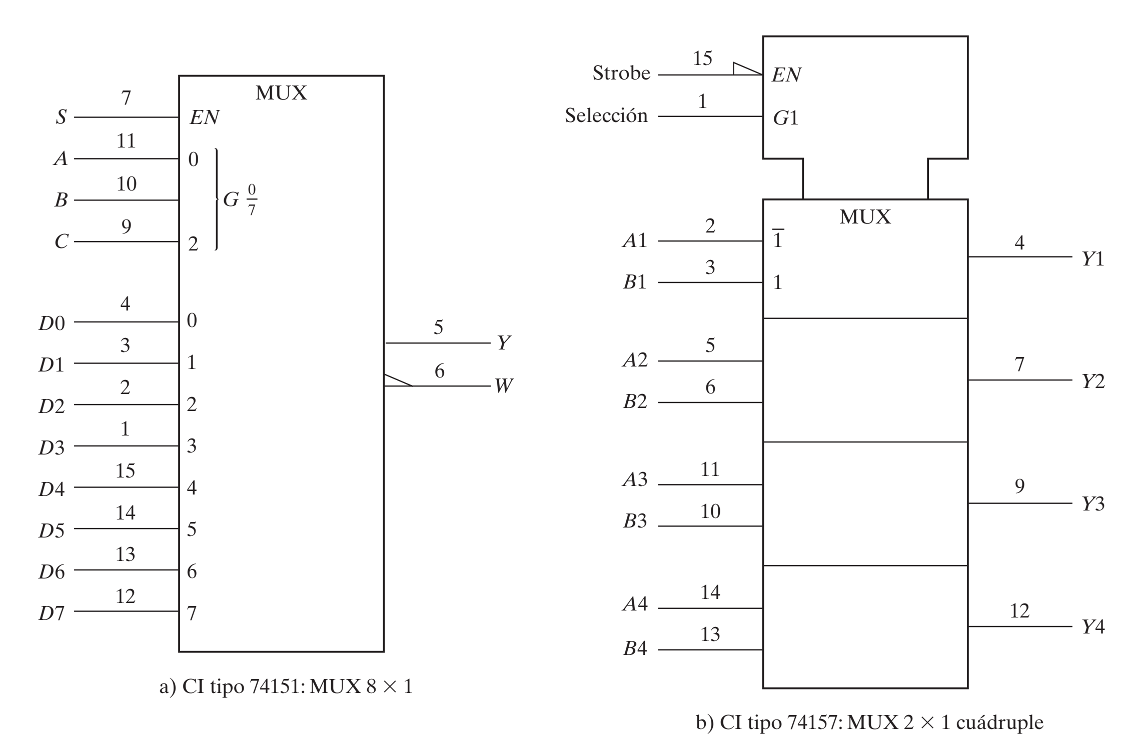


FIGURA 12-7
Símbolos gráficos para multiplexores

bloque forman parte de la notación estándar, pero los que están afuera son definidos por el usuario. La tabla de función del CI 74151 está en la figura 11-9. La dependencia AND se marca con $G\frac{0}{7}$ y está asociada a las entradas indicadas por la llave. Estas entradas tienen pesos de 0, 1 y 2. En realidad, son lo que hemos llamado entradas de selección. Las ocho entradas de datos están marcadas con números del 0 al 7. El peso neto de las entradas activas asociadas al símbolo G especifica el número de la entrada de datos que está activa. Por ejemplo, si las entradas de selección $CBA = 110$, las entradas 1 y 2 asociadas a G estarán activas. Esto da un valor numérico para la dependencia AND de $2^2 + 2^1 = 6$, lo que activa a $G6$. Puesto que se forma un AND entre $G6$ y la entrada de datos número 6, esta entrada está activa. Por tanto, la salida será igual a la entrada de datos D_6 siempre que la entrada de habilitación esté activa.

La figura 12-7b) representa el CI de cuádruple multiplexor 2×1 tipo 74157, cuya tabla de función se da en la figura 11-17. Las entradas de habilitación y selección son comunes a los cuatro multiplexores. Esto se indica en la notación estándar con el recuadro en forma de "T" en la parte superior del diagrama, que representa un *bloque de control común*. Las entradas a un bloque de control común controlan todas las secciones inferiores del diagrama. La entrada de habilitación común EN está activa cuando está en el estado de bajo nivel. La dependencia AND, $G1$, determina qué entrada está activa en cada sección de multiplexor. Cuando $G1 = 0$, las entradas A marcadas con $\overline{1}$ están activas. Cuando $G1 = 1$, las entradas B marcadas con 1 están activas. Las entradas activas se aplican a las salidas correspondientes si EN está activa. Observe que los símbolos $\overline{1}$ y 1 se marcan únicamente en la sección superior, en vez de repetirse en cada sección.

12-5 SÍMBOLOS PARA FLIP-FLOPS

En la figura 12-8 se muestran símbolos gráficos estándar para diferentes tipos de flip-flops. Un flip-flop se representa con un bloque rectangular con las entradas a la izquierda y las salidas a la derecha. Una salida designa el estado normal del flip-flop y la otra, la que tiene el pequeño símbolo circular de negación (o el indicador de polaridad), designa la salida complemento. Los símbolos gráficos distinguen tres tipos de flip-flops: el latch D, cuya construcción interna se muestra en la figura 6-5; el flip-flop amo-esclavo, que se observa en la figura 6-9; y el flip-flop disparado por flanco, que se introdujo en la figura 6-12. El símbolo gráfico para el latch D o flip-flop D tiene las entradas D y C indicadas dentro del bloque. El símbolo gráfico para el flip-flop JK tiene las entradas J, K y C indicadas en su interior. Las notaciones $C1$, $1D$, $1J$ y $1K$ son ejemplos de dependencias de control. La entrada en $C1$ controla la entrada $1D$ en un flip-flop D y las entradas $1J$ y $1K$ en un flip-flop JK.

El latch D no tiene más símbolos además de las entradas $1D$ y $C1$. El flip-flop disparado por flanco tiene un símbolo con forma de punta de flecha delante de la dependencia de control $C1$ para designar una entrada dinámica. El símbolo indicador dinámico significa que el flip-flop responde a la transición de borde positivo de los pulsos de reloj de entrada. Un círculo pequeño afuera del bloque, sobre el indicador dinámico, designa una transición de borde negativo para disparar el flip-flop. El amo-esclavo se considera un flip-flop disparado por pulso y se indica como tal con un símbolo de L invertida adelante de las salidas. Esto indica que la señal de salida cambia en el flanco descendente del pulso. Observe que el flip-flop amo-esclavo se dibuja sin el indicador dinámico.

Los flip-flops disponibles en paquetes de circuitos integrados tienen entradas especiales para establecer y restablecer el flip-flop asincrónicamente. Estas entradas suelen llamarse esta-

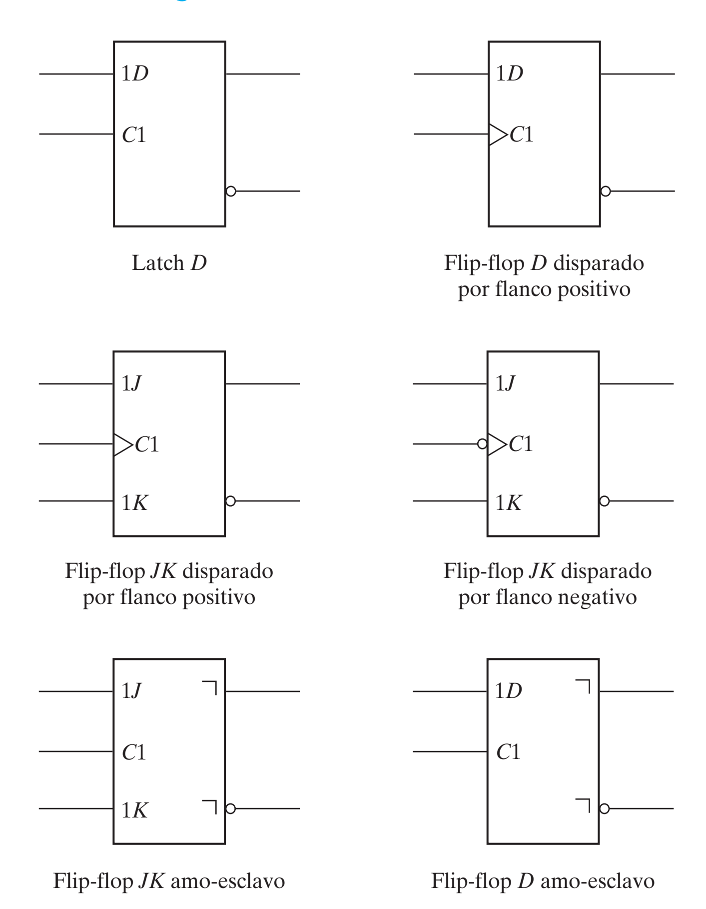


FIGURA 12-8
Símbolos gráficos estándar para flip-flops

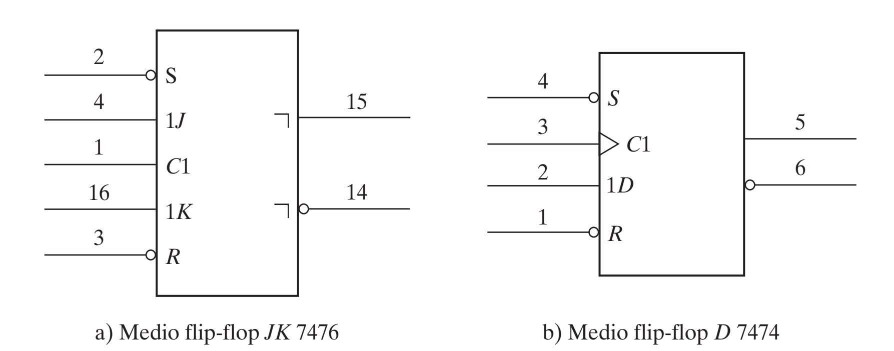


FIGURA 12-9
Flip-flops en CI con establecimiento y restablecimiento directos

blecimiento directo y restablecimiento directo. Afectan la salida en el nivel negativo de la señal, sin necesidad de un reloj. El símbolo gráfico de un flip-flop *JK* amo-esclavo con establecimiento y restablecimiento directos se muestra en la figura 12-9a). Las notaciones *C*1, 1*J* y 1*K* representan dependencia de control, e indican que la entrada de reloj en *C*1 controla las entradas 1*J* y 1*K*. *S* y *R* no tienen un 1 antes de las letras, así que no son controladas por el reloj en *C*1. Las entradas *S* y *R* tienen una burbuja en las líneas de entrada para indicar que están activas cuando están en el nivel de 0 lógico. La tabla de función del flip-flop 7476 se muestra en la figura 11-12.

El símbolo gráfico de un flip-flop *D* disparado por flanco positivo, con establecimiento y restablecimiento directos, se ilustra en la figura 12-9b). La transición de borde positivo del reloj en la entrada *C*1 controla la entrada 1*D*. Las entradas *S* y *R* son independientes del reloj. Éste es el CI tipo 7474, cuya tabla de función se da en la figura 11-13.

12-6 SÍMBOLOS PARA REGISTROS

El símbolo gráfico estándar para un registro es equivalente al símbolo empleado para un grupo de flip-flops con una entrada de reloj común. La figura 12-10 muestra el símbolo gráfico estándar del CI tipo 74175, que consiste en cuatro flip-flops *D* con entradas de reloj y de despeje comunes. La entrada de reloj *C*1 y la entrada de despeje *R* aparecen en el bloque de control común. Las entradas a dicho bloque se conectan a cada uno de los elementos de las secciones

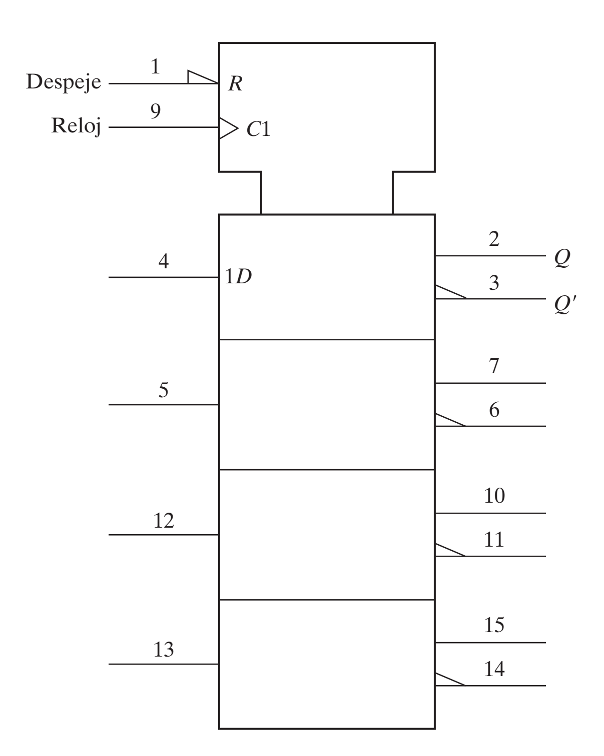


FIGURA 12-10
Símbolo gráfico de un registro de 4 bits, CI tipo 74175

inferiores del diagrama. La notación *C*1 es la dependencia de control que controla todas las entradas 1*D*. Así, cada flip-flop se dispara con la entrada de reloj común. El símbolo de entrada dinámica asociado a *C*1 indica que los flip-flops se disparan con el borde positivo del reloj de entrada. La entrada *R* común restablece todos los flip-flops cuando está en el estado de nivel bajo. El símbolo 1*D* sólo se escribe una vez en la sección superior, en lugar de repetirse en cada sección. Las salidas de complemento de los flip-flops de este diagrama se marcan con el símbolo de polaridad, no con el de negación.

El símbolo gráfico estándar para un registro de desplazamiento con carga paralela se aprecia en la figura 12-11. Éste es el CI tipo 74195, cuya tabla de función se encuentra en la figura 11-16. El símbolo calificador para un registro de desplazamiento es *SRG* seguido de un número que especifica el número de etapas. Por ejemplo, *SRG*4 denota un registro de desplazamiento de cuatro bits. El bloque de control común tiene dos dependencias de modo, *M*1 y *M*2, para las operaciones de desplazamiento y de carga, respectivamente. Vemos que el CI tiene una sola entrada rotulada *SH*/*LD* (desplazamiento/carga), que se divide en dos líneas para indicar los dos modos. *M*1 está activo cuando la entrada *SH*/*LD* es alta, y *M*2 está activo cuando *SH*/*LD* es baja. *M*2 se reconoce como activo-bajo por el indicador de polaridad en su línea de entrada. Note la convención en esta simbología: debemos reconocer que en la terminal 9 en realidad hay una sola entrada, pero se divide en dos partes para asignarle los dos modos, *M*1 y *M*2. La dependencia de control *C*3 es para la entrada de reloj. El símbolo dinámico en la entrada *C*3 indica que los flip-flops se disparan con el borde positivo del reloj. El símbolo /1→ después de *C*3 indica que el registro desplaza a la derecha (en la dirección hacia abajo) cuando el modo *M*1 está activo.

Las cuatro secciones bajo el bloque de control común representan los cuatro flip-flops. El flip-flop *QA* tiene tres entradas: dos asociadas a la operación en serie (desplazamiento) y una

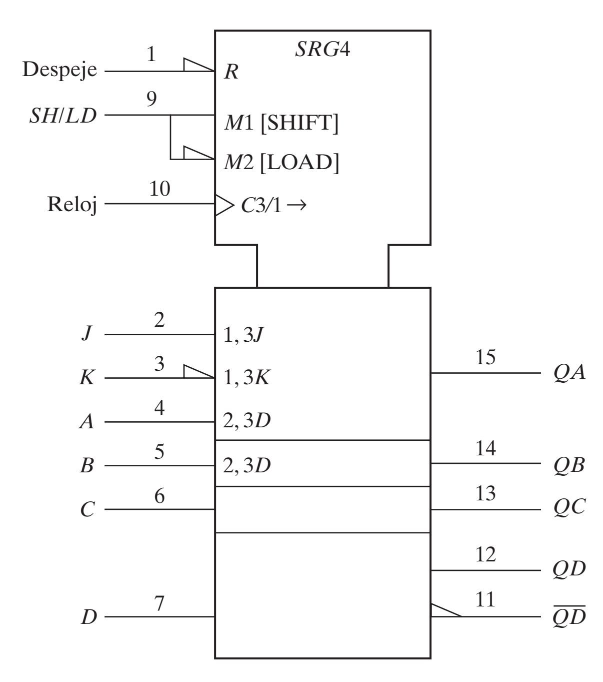


FIGURA 12-11
Símbolo gráfico de un registro de desplazamiento con carga paralela, CI tipo 74195

asociada a la operación en paralelo (carga). El rótulo de la entrada en serie 1, 3*J* indica que la entrada *J* del flip-flop *QA* está activa cuando *M*1 (desplazamiento) está activo y *C*3 tiene una transición positiva de reloj. La otra entrada en serie con el rótulo 1, 3*K* tiene un símbolo de polaridad en su línea de entrada que corresponde al complemento de la entrada *K* de un flip-flop *JK*. La tercera entrada de *QA* y las entradas de los otros flip-flops son para los datos de entrada en paralelo. Cada entrada se designa con el rótulo 2, 3*D*. El 2 se refiere a *M*2 (carga) y el 3 es por el reloj *C*3. Si la entrada de la terminal número 9 está en el nivel bajo, *M*1 está activo y una transición positiva del reloj en *C*3 causa una transferencia en paralelo desde las cuatro entradas *A*-*D* hacia los cuatro flip-flops, *QA*-*QD*. Advierta que la entrada paralela sólo se rotula en las secciones primera y segunda. Se supone que está en las otras dos secciones de abajo.

La figura 12-12 muestra el símbolo gráfico para el registro de desplazamiento bidireccional con carga paralela, CI tipo 74194. La tabla de función para este CI se incluye en la figura 11-19. El bloque de control común muestra una entrada *R* para restablecer todos los flip-flops en 0 asincrónicamente. La selección de modo tiene dos entradas, y la dependencia de modo *M* puede adoptar valores binarios de 0 a 3. Esto se indica con el símbolo $M\frac{0}{3}$, que representa *M*0, *M*1, *M*2, *M*3 y es similar a la notación para la dependencia *G* en los multiplexores. El símbolo asociado al reloj es

$$C4/1 \rightarrow \quad /2 \leftarrow$$

*C*4 es la dependencia de control para el reloj. El símbolo $/1\rightarrow$ indica que el registro desplaza a la derecha (hacia abajo en este caso) cuando el modo es *M*1 ($S_1S_0 = 01$). El símbolo $/2 \leftarrow$ indica que el registro desplaza a la izquierda (hacia arriba en este caso) cuando el modo es *M*2 ($S_1S_0 = 10$). Se obtienen las direcciones a la derecha y a la izquierda cuando la página se gira 90 grados en sentido contrario a las manecillas del reloj.

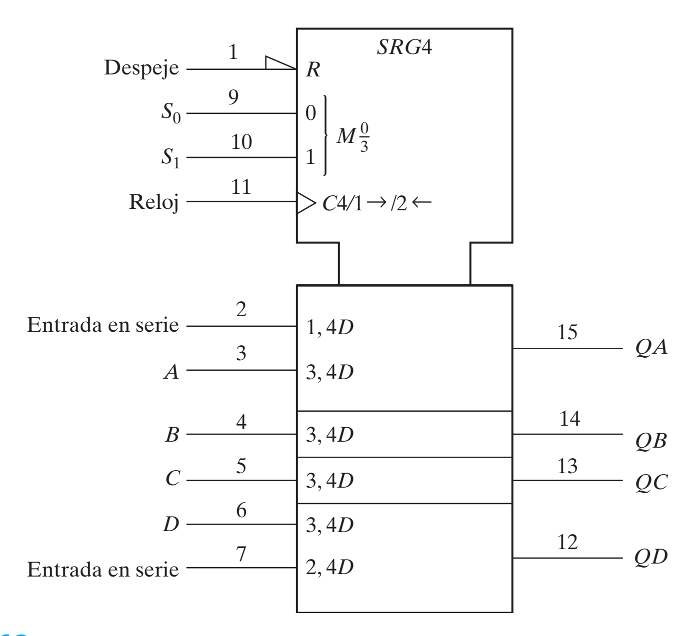


FIGURA 12-12
Símbolo gráfico para un registro de desplazamiento bidireccional con carga paralela, CI tipo 74194

Las secciones que están abajo del bloque de control común representan los cuatro flip-flops. El primero tiene una entrada en serie para desplazamiento a la derecha, rotulada con 1,4*D* (modo *M*1, reloj *C*4, entrada *D*). El último flip-flop tiene una entrada en serie para desplazamiento a la izquierda, rotulada con 2,4*D* (modo *M*2, reloj *C*4, entrada *D*). Los cuatro flip-flops tienen una entrada en paralelo rotulada con 3,4*D* (modo *M*3, reloj *C*4, entrada *D*). Por tanto, *M*3 ($S_1S_0 = 11$) es para la carga paralela. El modo restante, *M*0 ($S_1S_0 = 00$) no afecta las salidas porque no está incluido en los rótulos de entrada.

12-7 SÍMBOLOS PARA CONTADORES

El símbolo gráfico estándar para un contador binario con rizo se presenta en la figura 12-13. El símbolo calificador de un contador con rizo es *RCTR*. La designación *DIV*2 representa el circuito de dividir entre 2 que se obtiene del flip-flop *QA* solo. La designación *DIV*8 es para el contador de dividir entre 8 que se obtiene de los otros tres flip-flops. El diagrama representa el CI tipo 7493, cuyo diagrama de circuitos interno se aprecia en la figura 11-2. El bloque de control común tiene una compuerta AND interna, con entradas *R*1 y *R*2. Cuando ambas entradas son 1, el contenido del contador se pone en ceros. Esto se indica con el símbolo $CT = 0$. Puesto que la entrada de conteo no alimenta las entradas de reloj de todos los flip-flops, no lleva el rótulo *C*1, sino que se usa el símbolo + para indicar una operación de conteo ascendente. El símbolo dinámico cerca del +, junto con el símbolo de polaridad a lo largo de la línea de entrada, indica que una transición de borde negativo de la señal de entrada afecta el conteo. Los agrupamientos de bits de 0 a 2 en la salida representan valores de potencias de 2 para los pesos. Así, 0 representa el valor de $2^0 = 1$, y 2 representa el valor de $2^2 = 4$.

El símbolo gráfico estándar para el contador de 4 bits con carga paralela, CI tipo 74161, se muestra en la figura 12-14. El símbolo calificador para un contador sincrónico es *CTR* seguido del símbolo *DIV*16 (dividir entre 16), que da la longitud de ciclo del contador. Hay una

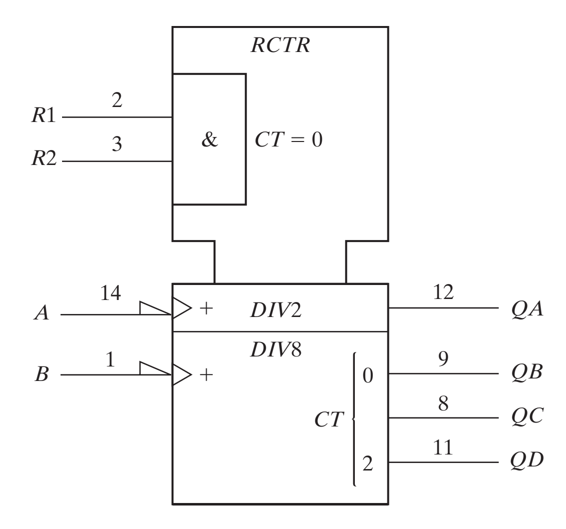


FIGURA 12-13
Símbolo gráfico para el contador de rizo, CI tipo 7493

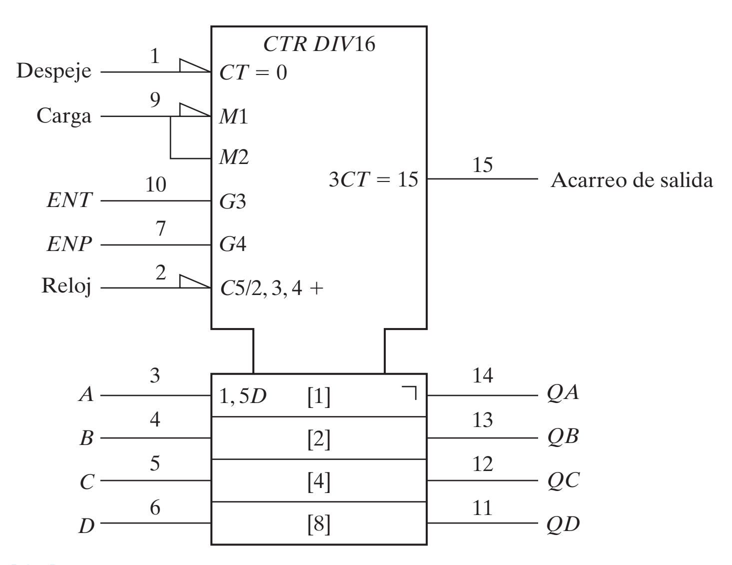


FIGURA 12-14
Símbolo gráfico para un contador binario de 4 bits con carga paralela, CI tipo 74161

sola entrada de carga en la terminal 9 que se divide en los dos modos, *M*1 y *M*2. *M*1 está activo cuando la entrada de carga en la terminal 9 está baja; *M*2 está activo cuando esa entrada está alta. *M*1 se reconoce como activa-baja por el indicador de polaridad a lo largo de su línea de entrada. Las entradas de habilitación de conteo utilizan las dependencias *G*. *G*3 está asociada a la entrada *T*, y *G*4, a la entrada *P* de la habilitación de conteo. El rótulo asociado al contador es

$$C\,5/2,\,3,\,4\ \ +$$

Esto significa que el circuito cuenta hacia arriba (el símbolo +) cuando *M*2, *G*3 y *G*4 están activas (carga = 1, *ENT* = 1 y *ENP* = 1) y el reloj en *C*5 pasa por una transición positiva. Esta condición se especifica en la tabla de función del 74161 de la figura 11-15. Las entradas paralelas tienen el rótulo 1,5*D*, lo que significa que las entradas *D* están activas cuando *M*1 está activo (carga = 0) y el reloj pasa por una transición positiva. El acarreo de salida se rotula con

$$3CT\ =\ 15$$

Esto se interpreta como que el acarreo de salida está activo (igual a 1) si *G*3 está activo (*ENT* = 1) y el contenido (*CT*) del contador es 15 (1111 binario). Observe que las entradas tienen un símbolo de *L* invertida, lo que indica que todos los flip-flops son del tipo amo-esclavo. El símbolo de polaridad en la entrada *C*5 denota un pulso invertido para la entrada de reloj. Esto implica que el amo se dispara con la transición negativa del pulso de reloj y el esclavo cambia de estado con la transición positiva. Por tanto, la salida cambia con la transición positiva del pulso de reloj. Cabe señalar que el CI tipo 74LS161 (versión Schottky de baja potencia) tiene flip-flops disparados por flanco positivo.

12-8 SÍMBOLO PARA RAM

El símbolo gráfico estándar para la memoria de acceso aleatorio (RAM) 74189 aparece en la figura 12-15. Los números 16 × 4 que siguen al símbolo calificador RAM indican el número de palabras y el número de bits por palabra. El bloque de control común se muestra con cuatro líneas de dirección y dos entradas de control. Cada bit de la palabra aparece en una sección aparte con una línea de datos de entrada y una de salida. Se usa la dependencia de dirección *A* para identificar las entradas de dirección de la memoria. Las entradas y salidas de datos afectadas por la dirección están rotuladas con la letra *A*. Los agrupamientos de bits 0 a 3 proporcionan la dirección binaria que va desde *A*0 hasta *A*15. El triángulo invertido indica salidas de tres estados. El símbolo de polaridad especifica la inversión de las salidas.

El funcionamiento de la memoria se especifica con la notación de dependencia. El símbolo gráfico de RAM usa cuatro dependencias: *A* (dirección), *G* (AND), *EN* (habilitar) y *C* (control). La entrada *G*1 debe considerarse en AND con 1*EN* y 1*C*2 porque *G*1 tiene un 1 después de la letra *G* y las otras dos tienen un 1 en su rótulo. Se usa la dependencia *EN* para identificar una entrada de habilitación que controla las salidas de datos. La dependencia *C*2 controla las entradas, como indica el rótulo 2*D*. Así pues, para una operación de escritura, tenemos la dependencia *G*1 y 1*C*2 (*CS* = 0), la dependencia *C*2 y 2*D* (*WE* = 0), y la dependencia *A*, que especifica la dirección binaria en las cuatro entradas de dirección. Para una operación de lectura, tenemos las dependencias *G*1 y 1*EN* (*CS* = 0, *WE* = 1) y la dependencia *A* para las salidas. La interpretación de estas dependencias da como resultado el funcionamiento de la memoria descrito en la tabla de función de la figura 11-18.

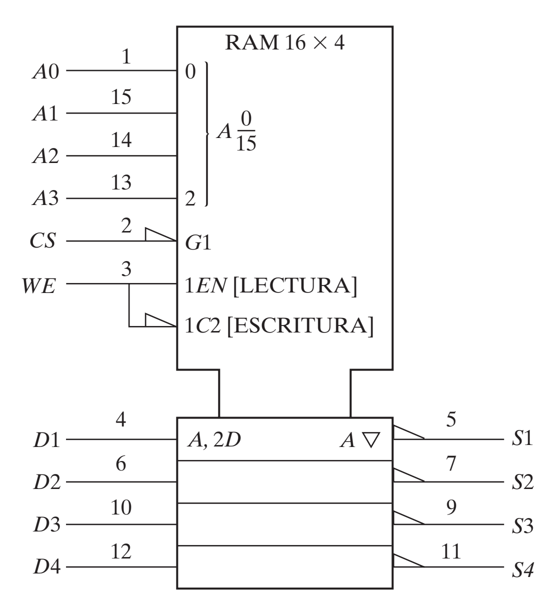


FIGURA 12-15
Símbolo gráfico para RAM 16 × 4, CI tipo 74189

PROBLEMAS

12-1 La figura 11-1 ilustra diversos circuitos de integración a pequeña escala con su asignación de terminales. Utilizando esta información, dibuje los símbolos gráficos rectangulares para los CI 7400, 7404 y 7486.

12-2 Describa lo siguiente con sus propias palabras:

a) Lógica positiva y negativa.
b) Activa-alta y activa-baja.
c) Indicador de polaridad.
d) Indicador dinámico.
e) Notación de dependencia.

12-3 Dé un ejemplo de símbolo gráfico que tenga las tres dependencias booleanas, *G*, *V* y *N*. Dibuje la interpretación equivalente.

12-4 Dibuje el símbolo gráfico de un decodificador BCD a decimal. Es similar a un decodificador con 4 entradas y 10 salidas.

12-5 Dibuje el símbolo gráfico de un decodificador de binario a octal con tres entradas de habilitación, $E1$, $E2$ y $E3$. El circuito se habilita si $E1 = 1$, $E2 = 0$ y $E3 = 0$ (suponiendo lógica positiva).

12-6 Dibuje el símbolo gráfico de un CI con dos multiplexores de 4 líneas a 1 con entradas de selección comunes y una entrada de habilitación aparte para cada multiplexor.

12-7 Dibuje el símbolo gráfico para los siguientes flip-flops:

a) Flip-flop *D* disparado por borde negativo.
b) Flip-flop *RS* amo-esclavo.
c) Flip-flop *T* disparado por borde positivo.

12-8 Explique la función del bloque de control común utilizado con los símbolos gráficos estándar.

12-9 Dibuje el símbolo gráfico de un registro de 4 bits con carga paralela empleando el rótulo $M1$ para la entrada de carga y $C1$ para el reloj.

12-10 Explique todos los símbolos empleados en el diagrama gráfico estándar de la figura 12-12.

12-11 Dibuje el símbolo gráfico de un contador binario sincrónico ascendente-descendente con entrada de modo (para ascendente o descendente) y entrada de habilitación de conteo con dependencia *G*. Muestre los acarreos de salida para el conteo ascendente y el conteo regresivo.

12-12 Dibuje el símbolo gráfico de una RAM de 256×1. Incluya el símbolo para salidas de tres estados.

REFERENCIAS

1. 1984. *IEEE Standard Graphic Symbols for Logic Functions*. (ANSI/IEEE Std. 91-1984). Nueva York: Institute of Electrical and Electronics Engineers.
2. Kampel, I. 1985. *A Practical Introduction to the New Logic Symbols*. Boston: Butterworth.
3. Mann, F. A. 1984. *Explanation of New Logic Symbols*. Dallas: Texas Instruments.
4. 1985. *The TTL Data Book*, volumen 1. Dallas: Texas Instruments.

Respuestas a problemas selectos

CAPÍTULO 1

1-2 **a)** 32,768 **b)** 67,108,864 **c)** 6,871,947,674

1-4 $(4310)_5 = 580$ $(198)_{12} = 260$

1-5 **a)** 6 **b)** 8 **c)** 11

1-6 8

1-7 22.3125 (los tres)

1-9 64276 octal

1-12 **a)** 10000 y 110111 **b)** 62 y 958

1-19 **a)** 010627 **b)** 009025 **c)** 990975 **d)** 989373

1-23

6	3	1	1	Decimal
0	0	0	0	0
0	0	0	1	1
0	0	1	1	2
0	1	0	0	3
0	1	1	0	4 (*o* 0101)
0	1	1	1	5
1	0	0	0	6
1	0	1	0	7 (*o* 1001)
1	0	1	1	8
1	1	0	0	9

1-28 Jane Doe

1-30 62 + 32 = 94 caracteres imprimibles

1-31 El bit 6 desde la derecha

1-32 **a)** 897 **b)** 564 **c)** 871 **d)** 2,199

CAPÍTULO 2

2-2 **a)** x **b)** x **c)** y **d)** 0

2-3 **a)** B **b)** $z(x + y)$ **c)** $x'y'$ **d)** $x(w + z)$ **e)** 0

2-4 **a)** $AB + C'$ **b)** $x + y + x$ **c)** B **d)** $A'(B + C'D)$

2-6 **a)** $xy + x'y'$

2-8 $F(x, y, z) = \sum(1, 4, 5, 6, 7)$

2-9 **a)** 10001100 **c)** 00100011 **d)** 01010010

2-11 **b)** $(x' + y')' + (x + y)' + (y + z')'$

2-12 $T_1 = A'(B' + C')$
$T_2 = A + BC = T_1'$

2-14 **a)** $\sum(3, 5, 6, 7) = \prod(0, 1, 2, 4)$

2-15 **c)** $F = y'z + y(w + x)$

2-16 $\sum(1, 3, 5, 7, 9, 11, 13, 15) = \prod(0, 2, 4, 6, 8, 10, 12, 14)$

2-19 **a)** $AB + BC = (A + C)B$ **b)** $x' + y + z'$

CAPÍTULO 3

3-1 **a)** $xy + x'z'$ **b)** $C' + A'B$ **c)** $a' + bc$ **d)** $xy + xz + yz$

3-2 **a)** $x'y' + xz$ **b)** $y + x'z$

3-3 **a)** $xy + x'z'$ **b)** $x' + yz$ **c)** $C' + A'B$

3-4 **a)** y **b)** $BCD + A'BD'$ **c)** $ABD + ABC + CD$ **d)** $wx + w'x'y$

3-5 **a)** $xz' + w'y'z + wxy$ **d)** $BD + B'D' + A'B$ o $BD + B'D' + A'D'$

3-6 **a)** $B'D' + A'BD + ABC'$ **b)** $xy' + x'z + wx'y$

3-7 **a)** $x'y + z$ **c)** $AC + B'D' + A'BD + B'C$ (o CD)

3-8 **a)** $F(x, y, z) = \sum(3, 5, 6, 7)$ **b)** $F(A, B, C, D) = \sum(1, 3, 5, 9, 12, 13, 14)$

3-9 **a)** Esenciales: xz y $x'z'$; no esenciales: $w'x$ y $w'z'$
b) $F = B'D' + AC + A'BD + (CD$ o $B'C)$

3-10 **c)** $F = BC' + AC + A'B'D$
Esenciales: BC', AC
No esenciales: AB, $A'B'D$, $B'CD$, $A'C'D$

3-11 **a)** $F = A'B'D' + AD'E + B'C'D'$

3-12 **b)** $F = (A + D')(B' + D')$

3-13 **a)** $F = xy + z' = (x + z')(y + z')$

3-15 **b)** $F = B'D' + CD' + ABC'D = \sum(0, 2, 6, 8, 10, 13, 14)$

3-17 $F' = BD + BC + AC$

3-19 **a)** $F = (w + z')(x' + z')(w' + x' + y')$

3-30 $F = (A \oplus B)(C \oplus D)$

3-35 Línea 1: No se permiten guiones, usar subraya: Ejm_3.
Falta punto y coma (;) al final.

Línea 2: "inputs" debe ser "input" (sin s final).
Cambiar última coma (,) a punto y coma (;).

Línea 3: No mayúsculas en "Output", cambiar a "output".

Línea 4: "A" no puede ser salida (se definió como entrada).
"D" no puede ser entrada (se definió como salida).

Línea 5: Demasiados elementos para la compuerta NOT (sólo se permiten dos).

Línea 6: OR debe estar en minúscula: cambiar a "or".

Línea 7: Quitar punto y coma (no debe ir después de "endmodule").

CAPÍTULO 4

4-1 **a)** $F_1 = A + B'C + BD' + B'D$
$F_2 = A'B + D$

4-2 $F = ABC + A'D$
$G = ABC + A'D'$

4-3 **b)** 1024 filas y 14 columnas

4-4 $F = x'y' + x'z'$

4-6 $F = xy + xz + yz$

4-7 $w = A \quad x = A \oplus B \quad y = x \oplus C \quad z = y \oplus D$

4-8 $w = AB + AC'D'$

4-10 Entradas: A, B, C, D; Salidas: w, x, y, z
$z = D$
$y = C \oplus D$
$x = B \oplus (C + D)$
$w = A \oplus (B + C + D)$

4-12 **b)** $D = x \oplus y \oplus z$
$B = x'y + x'z + yz$

4-13

	Suma	C	V
a)	1101	0	1
b)	0001	1	1
c)	0100	1	0
d)	1011	0	1
e)	1111	0	0

4-14 60 ns

4-18
$w = A'B'C'$
$x = B \oplus C$
$y = C$
$z = D'$

4-22
$w = AB + ACD$
$x = B'C' + B'D' + BCD$
$y = C'D + CD'$
$z = D'$

4-28
$F_1 = \sum(0, 5, 7)$
$F_2 = \sum(2, 3, 4)$
$F_3 = \sum(1, 6, 7)$

4-29
$x = D_0'D_1'$
$y = D_0'D_1 + D_0'D_2'$

4-34 $F(A, B, C, D) = \sum(1, 6, 7, 9, 10, 11, 12)$

4-35
Cuando $AB = 00$, $F = D$
Cuando $AB = 01$, $F = (C + D)'$
Cuando $AB = 10$, $F = CD$
Cuando $AB = 11$, $F = 1$

4-39

```
module Compare (A,B,Y);
input [3:0] A,B; //Entradas de datos de 4 bits.
output [5:0] Y;  //Salida de comparador de 6 bits.
reg [5:0] Y;     //EQ,NE,GT,LT,GE,LE
always @ (A or B)
      if (A==B)      Y = 6'b100011; //EQ,GE,LE
 else if (A < B)     Y = 6'b010101; //NE,LT,LE
      else           Y = 6'b011010; //NE,GT,GE
endmodule
```

4-42 c)

```
module Convrt_bhv (BCD, EXS3);
input [4:1] BCD;
output [4:1] EXS3;
reg [4:1] EXS3;
   always @ (BCD)
     EXS3 = BCD + 4'b0011;
endmodule
```

CAPÍTULO 5

5-4 **b)** $PQ' + NQ$

5-7 $S = x \oplus y \oplus Q$

$Q(t + 1) = xy + xQ + yQ$

5-8 Un contador con la sucesión repetida 00, 01, 10

5-9 **a)** $A(t + 1) = xA' + AB$

$B(t + 1) = xB' + A'B$

5-10 **c)** $A(t + 1) = xB + x'A + yA + y'A'B'$

$B(t + 1) = xA'B' + x'A'B + y'A'B$

5-11

Estado actual:	00	00	01	00	01	11	00	01	11	10	00	01	11	10	10
Entrada:	0	1	0	1	1	0	1	1	1	0	1	1	1	1	0
Salida:	0	0	1	0	0	1	0	0	0	1	0	0	0	0	1
Siguiente estado:	00	01	00	01	11	00	01	11	10	00	01	11	10	10	00

5-12

Estado actual	Siguiente Estado 0	Siguiente Estado 1	Salida 0	Salida 1
a	f	b	0	0
b	d	a	0	0
d	g	a	1	0
f	f	b	1	1
g	g	d	0	1

5-13 **a)** Estado: $a\ f\ b\ c\ e\ d\ g\ h\ g\ g\ h\ a$

Entrada: 0 1 1 1 0 0 1 0 0 1 1

Salida: 0 1 0 0 0 1 1 1 0 1 0

b) Estado: $a\ f\ b\ a\ b\ d\ g\ d\ g\ g\ d\ a$

Entrada: 0 1 1 1 0 0 1 0 0 1 0

Salida: 0 1 0 0 0 1 1 1 0 1 0

5-15 $D_Q = Q'J + QK'$

5-16 $D_A = Ax' + Bx$

$D_B = A'x + Bx'$

5-18 $J_A = K_A = (Bx + B'x')E$

$J_B = K_B = E$

5-19 **a)** $D_A = A'B'x$

$D_B = A + C'x' + BCx$

$D_C = Cx' + Ax + A'B'x'$

$y = A'x$

5-23 **a)** RegA = 125, RegB = 125

b) RegA = 125, RegB = 30

5-26 $Q(t + 1) = JQ' + K'Q$
Cuando $Q = 0, Q(t + 1) = J$
Cuando $Q = 1, Q(t + 1) = K'$

```
always @ (posedge CLK)
   if (Q==0) Q = J;
   else Q = ~K;
```

5-30 Dos flip-flops E y Q con CLK.
Compuerta AND con entradas A y B va a la entrada D de E.
Compuerta OR con entradas E y C va a la entrada D de Q.
Si los enunciados usan asignación bloqueadora con el símbolo (=), el flip-flop Q deberá mostrarse con su entrada D conectada, no a C + E, sino a C + AB (una compuerta AND y una compuerta OR).

CAPÍTULO 6

6-4 1110; 0111; 1011; 1101; 0110; 1011

6-8 $A = 0010, 0001, 1000, 1100.$ Acarreo $= 1, 1, 1, 0$

6-9 **b)** $J_Q = x'y; K_Q = (x' + y)'$

6-14 **a)** 4; **b)** 9; **c)** 10

6-15 50 ns; 20 MHz

6-16 $1010 \rightarrow 1011 \rightarrow 0100$
$1100 \rightarrow 1101 \rightarrow 0100$
$1110 \rightarrow 1111 \rightarrow 0000$

6-17 $D_{A0} = A_0 \oplus E$
$D_{A1} = A_1 \oplus (A_0E)$
$D_{A2} = A_2 \oplus (A_1A_0E)$
$D_{A3} = A_3 \oplus (A_2A_1A_0E)$

6-19 **b)** $D_{Q1} = Q_1'$
$D_{Q2} = Q_2Q_1' + Q_8'Q_2'Q_1$
$D_{Q4} = Q_4Q_1' + Q_4Q_2' + Q_4'Q_2'Q_1$
$D_{Q8} = Q_8Q_1' + Q_4Q_2Q_1$

6-21 $J_{A0} = LI_0 + L'C$
$K_{A0} = LI_0' + L'C$

6-24 $T_A = A \oplus B$
$T_B = B \oplus C$
$T_C = AC + A'C'$ (sin autoinicio)
$= AC + A'B'C$ (con autoinicio)

6-26 Usar un contador de 2 bits

6-28 $D_A = A \oplus B$
$D_B = AB' + C$
$D_C = A'B'C'$

6-34

```
module Shiftreg (SI,SO,CLK);
  input SI,CLK;
  output SO;
  reg [3:0] Q;
  assign SO = Q[0];
  always @ (posedge CLK)
    Q = {SI,Q[3:1]};
endmodule
```

6-36 **a)**

```
module updown (Up,Down,Load,IN,OUT,CLK);
  input Up,Down,Load,CLK;
  input [3:0] IN;
  output [3:0] OUT;
  reg [3:0] OUT;
  always @ (posedge CLK)
    if (Load) OUT = IN;
    else if (Up) OUT = OUT + 4'b0001;
    else if(Down)OUT = OUT -  4'b0001;
    else OUT = OUT;
endmodule
```

CAPÍTULO 7

7-2 **a)** 2^{13} **b)** 2^{31} **c)** 2^{26} **d)** 2^{21}

7-3 Dirección: 10 1101 0011 = 2D3 (hex)
Datos: 0000 1101 0111 1011 = 0E7B (hex)

7-7 **a)** Decodificadores 7 × 128, 256 compuertas AND **b)** $x = 46$; $y = 112$

7-8 **a)** 8 chips **b)** 18; 15 **c)** Decodificador 3 × 8

7-10 0001 1011 1011 1

7-11 101 110 011 001 010

7-12 **a)** 0101 1010; **b)** 1100 0110; **c)** 1111 0100

7-13 **a)** 6 **b)** 7 **c)** 7

7-14 **a)** 0101010

7-16 24 terminales

7-18 **a)** 256 × 8 **b)** 512 × 5 **c)** 1024 × 4 **d)** 32 × 7

7-20 Términos producto: yz', xz', $x'y'z$, xy', $x'y$, z

7-24 $A = yz' + xz' + x'y'z$
$B = x'y' + xy + yz$
$C = A + xyz$
$D = z + x'y$

CAPÍTULO 8

8-1 **a)** La transferencia y el incremento se efectúan en el mismo borde de reloj. Después de la transferencia, el contenido de R2 es 1 más que el contenido de R1.
b) Decrementar en 1 el contenido de R3.
c) Si (T1 = 1), transferir el contenido de R1 a R0. Si (T1 = 0 y T2 = 1), transferir el contenido de R2 a R0.

8-7 Diagrama de ASM:
T0: Estado inicial: si (S = 1), entonces (RA ← Entrada A, RB ← Entrada B, ir a T1).
T1: RA ← RA + (complemento a 2 de RB), Préstamo ← (complemento del acarreo), ir a T2.
T2: Si (Préstamo = 0) entonces ir a T0. Si (Préstamo = 1) entonces RA ← (complemento a 2 de RA), ir a T0.

8-8 T0: AR ← datos de entrada, BR ← datos de entrada.
T1: si (AR[15]) = 1 (bit de signo negativo) entonces CR ← AR (desplazado a la derecha, extensión de signo).
de lo contrario si (AR = 0) entonces (CR ← 0).
de lo contrario (positivo distinto de cero) entonces (Desbordamiento ← BR([15] ⊕ [14]), CR ← BR (desplazado a la izquierda).

8-9 $D_{T0} = T_2 + S'T_0$
$D_{T1} = ST_0 + (A_3A_4)'T_1$
$D_{T2} = A_3A_4T_1$

8-11 $D_A = A'B + Ax$
$D_B = A'B'x + A'By + xy$

8-14 Diagrama de ASM:
T0: (estado inicial) Si S = 0 regresar al estado T0,
Si (S = 1) entonces BR ← multiplicando, AR ← multiplicador, PR ← 0, ir a T1.
T1: (ver si AR es cero) Z = 1 si AR = 0,
si (Z = 1) entonces regresar a T0 (terminamos)
Si (Z = 0) entonces ir a T2.
T2: PR ← PR + BR, AR ← AR − 1, ir a T1.

8-15 $(2^n - 1)(2^n - 1) < (2^{2n} - 1)$ para $n \geq 1$

8-16 **a)** El tamaño máximo del producto es 32 bits, disponibles en los registros A y Q.
b) El contador P debe tener 5 bits para cargar 16 (binario 10000) inicialmente.
c) Z (detección de cero) se genera con una compuerta NOR de 5 entradas.

8-18 $2(n + 1)$t

8-19 MUX1: 0, 1, 1, Z′
MUX2: S, 0, 1, 0

8-21 **a)** $E = 1$ **b)** $E = 0$

8-22 A = 0110, B = 0010, C = 0000.

A * B = 1100	A	B = 0110	A && C = 0
A + B = 1000	A ∧ B = 0100	|A = 1
A − B = 0100	&A = 0	A < B = 0
~C = 1111	~|C = 1	A > B = 1
A & B = 0010	A || B = 1	A != B = 1

8-28

```
//Declarar todas las entradas, salidas y registros
      assign Z = ~|AR; //Z = 1 if AR = 0
      always @ (posedge CLK or negedge Clr)
         if (~Clr) pstate = T0;
         else pstate <= nstate;
      always @ (S or Z or pstate)
         case (pstate)
           T0: if (S) nstate = T1;
               else  nstate = T0;
           T1: if (Z) nstate = T0;
               else  nstate = T2;
           T2:    nstate = T1;
         endcase
      always @ (posedge CLK)
         case (pstate)
           T0: if (S)
                begin
                  AR <= Ain;
                  BR <= Bin;
                  PR <= 0;
                end
           T2:  begin
                  PR <= PR + BR;
                  AR <= AR - 8'b1;
                end
         endcase
```

CAPÍTULO 9

9-2 Sucesión de Y_1Y_2: 00, 00, 01, 11, 11, 01, 00.

9-3 **d)** Cuando la entrada es 01, la salida es 0. Cuando la entrada es 10, la salida es 1. Cuando la entrada asume una de las otras dos combinaciones, la salida conserva su valor anterior.

9-4 **c)**

	00	01	11	10
a	ⓐ, 0	*b*, 1	*c*, 1	*d*, 0
b	*a*, 0	ⓑ, 1	*c*, 1	ⓑ, 0
c	ⓒ, 1	*b*, 1	ⓒ, 1	*d*, 0
d	*c*, 1	*b*, 1	*c*, 1	ⓓ, 1

9-5 **c)** $Y_1 = x_1'x_2 + x_2y_1$
$Y_2 = x_2 + x_1y_2$
$z = x_1x_2y_1' + x_1y_2'$

9-10 $S = x_2x_2'$
$R = x_1'x_2$

9-13 **b)** Dos posibles tablas de transición:

	00	01	11	10
a	ⓐ, 0	b, –	–, –	e, –
b	ⓑ, 1	ⓑ, 1	–, –	d, –
d	a, –	ⓓ, 1	–, –	ⓓ, 1
e	ⓔ, 1	d, –	–, –	ⓔ, 1

	00	01	11	10
a	ⓐ, 0	b, –	–, –	b, –
b	c, –	ⓑ, 1	–, –	ⓑ, 0
c	ⓒ, 1	d, –	–, –	d, –
d	a, –	ⓓ, 1	–, –	ⓓ, 1

9-18 3a: $(a, b)(c, d)(e, f, g, h)$
3b: $(a, e, f)(b, j)(c, d)(g, h)(k)$

9-20 Añadir estados g y h a la asignación binaria.

	00	01	11	10
0	a	g	b	f
1	c	h	d	e

9-22 $F = A'D' + AC'D' + A'BC + A'CD'$

9-23 $Y = (x_1 + x_2')(x_2 + x_3)(x_1 + x_3)$

CAPÍTULO 10

10-1 Abanico de salida = 10; disipación de potencia = 18.75 mW; retardo de propagación = 3 ns; margen de ruido = 0.3 V

10-2 **a)** 1.058 V **b)** 0.82 V **c)** 0.238 V

10-3 $I_B = 0.44$ mA, $I_{CS} = 2.4$ mA

10-4 **a)** 2.4 mA **b)** 0.82 mA **c)** $2.4 + 0.82N$ **d)** 7.8 **e)** 7

10-5 **b)** 3.53 **c)** 2.585 mA **d)** 16 mA **e)** 300 Ω

10-9 **a)** 4.62 mA **b)** 4 mA

10-10 0.3 V

ÍNDICE

E

N

O

P

R

Acuerdo de licencia

SynaptiCAD
TestBencher Pro - DataSheet Pro - WaveFormer Pro - Timing Diagrammer Pro - VeriLogger Pro
Acuerdo de licencia de software
Léase antes de usar

Por favor, lea con atención esta licencia.

Nota: En este acuerdo, la palabra software se refiere al producto de software de SynaptiCAD del que usted ha adquirido la licencia.

Usted ha adquirido una licencia para utilizar alguno de los siguientes productos: software **TestBencher Pro, DataSheet Pro, WaveFormer Pro, VeriLogger Pro** o **Timing Diagrammer Pro**. El software pertenece y permanece en propiedad de SynaptiCAD, está protegido por derechos internacionales de autor y se transfiere al comprador original y cualquier poseedor subsecuente del software para su uso únicamente en los términos de la licencia explicitados más adelante. El hecho de abrir el empaque de **TestBencher Pro, DataSheet Pro**, **WaveFormer Pro, VeriLogger Pro** o **Timing Diagrammer Pro** y/o hacer uso de **TestBencher Pro, DataSheet Pro, WaveFormer Pro**, **VeriLogger Pro** o **Timing Diagrammer Pro** significa que usted ha aceptado estos términos. Si usted no está de acuerdo con todos estos términos y condiciones, por favor devuelva inmediatamente sin abrir el software y los manuales respectivos para un total reembolso.

Uso del software

- **SynaptiCAD** garantiza al comprador original (a quien en lo sucesivo se denomina “licenciado”) los derechos limitados para poseer y hacer uso del software y del Manual del Usuario (“software”) para sus propósitos. La persona que recibe la licencia está de acuerdo en que el software se utilizará solamente para sus propósitos internos y en que el software se instalará sólo en una computadora individual. Si el software se instala en un sistema en red o en una computadora conectada a un servidor de archivo u otro sistema que físicamente permita acceso compartido al software, el licenciado está de acuerdo en proveer métodos técnicos o de procedimiento a fin de prevenir el uso del software por más de un usuario.
- Puede hacerse una copia legible para una máquina SÓLO PARA PROPÓSITOS DE RESPALDO y, en ese caso, la copia deberá ostentar todas las advertencias del propietario y será etiquetada externamente para indicar que la copia de respaldo es propiedad de SynaptiCAD y que su uso está sujeto a la presente licencia.
- El uso del software por cualquier departamento, agencia o cualquier otra entidad del gobierno federal de Estados Unidos está limitado a los términos de la “Cláusula para usuarios de entidades gubernamentales” incluida más adelante.
- El licenciado puede transferir sus derechos bajo esta licencia, con la condición de que la parte a quien se transfieren los derechos esté de acuerdo con los términos y condiciones de esta licencia, y siempre que se haga una notificación por escrito a SynaptiCAD. En ese caso, el licenciado debe transferir o destruir todas las copias del software.
- A excepción de las condiciones expresadas en este acuerdo, el licenciado no habrá de modificar, alterar la ingeniería, descompilar, desensamblar, distribuir, sublicenciar, vender, rentar, arrendar, dar o transferir el software de ningún modo, ni por ningún medio, incluyendo las telecomunicaciones. El licenciado hará el mejor de sus esfuerzos y dará los pasos razonables para proteger el software del uso, copia o diseminación no autorizados y mantendrá intactas todas las advertencias del propietario.

GARANTÍA LIMITADA SynaptiCAD garantiza que el software está libre de defectos de fabricación por un periodo de noventa días a partir de la compra. Durante ese periodo, SynaptiCAD reemplazará sin costo cualquier software que presentara defectos y que sea devuelto a SynaptiCAD con gastos de envío prepagados. Este servicio es responsabilidad exclusiva de SynaptiCAD bajo esta garantía.

LIMITACIONES LOS DERECHOS DE LICENCIA DEL SOFTWARE NO INCLUYEN NINGUNA CONSIDERACIÓN QUE IMPLIQUE UN RIESGO PARA SYNAPTICAD, Y SYNAPTICAD NO SE HACE RESPONSABLE POR DAÑOS INCIDENTALES O A CONSECUENCIA DEL USO U OPERACIÓN O INHABILIDAD EN EL USO DEL SOFTWARE, AUN CUANDO LAS PARTES HAYAN SIDO ADVERTIDAS DE LAS POSIBILIDADES DE TALES DAÑOS. MÁS AÚN, EL LICENCIADO ESTÁ DE ACUERDO EN NO HACER ESTE TIPO DE RECLAMOS A SYNAPTICAD. EL LICENCIADO ASUME TODOS LOS RIESGOS DERIVADOS DE LOS RESULTADOS Y UTILIZACIÓN DEL SOFTWARE. LAS GARANTÍAS EXPRESADAS EN ESTA LICENCIA SON LAS ÚNICAS GARANTÍAS RECONOCIDAS POR SYNAPTICAD Y OCUPAN EL LUGAR DE CUALESQUIERA OTRAS GARANTÍAS, EXPRESAS O IMPLICADAS, INCLUYENDO PERO SIN LIMI-

TARSE A LAS GARANTÍAS IMPLICADAS EN LA COMERCIALIZACIÓN O CONVENIENCIA PARA UN PROPÓSITO PARTICULAR. ESTA GARANTÍA BRINDA A USTED DERECHOS LEGALES ESPECÍFICOS, Y PODRÁ TENER OTROS DERECHOS QUE VARÍAN DE UNA JURISDICCIÓN A OTRA. ALGUNAS JURISDICCIONES NO PERMITEN LA EXCLUSIÓN O LIMITACIÓN DE GARANTÍAS, ASÍ QUE, EN ESE CASO, NO SE APLICARÁN LAS SIGUIENTES LIMITACIONES O EXCLUSIONES.

Periodo

- Esta licencia es efectiva a partir del momento en que el licenciado recibe el software y continúa vigente hasta que el licenciado cese el uso del software y devuelva o destruya todas las copias del mismo, o cuando deje de cumplir con todos los términos de este acuerdo.

General

- Esta licencia es la declaración completa y exclusiva del acuerdo entre las partes. Cualquier estipulación de esta licencia declarada inválida por cualquier corte de la jurisdicción competente, será cumplida en el grado permisible y el resto de la licencia mantendrá, a pesar de ello, toda su fuerza y efecto. Esta licencia estará bajo el control de las leyes del Estado de Virginia y de los Estados Unidos de América.

Cláusula para usuarios de entidades del gobierno de los Estados Unidos
Ésta es una cláusula del Acuerdo de Licencia sobre el software **TestBencher Pro/DataSheet Pro/VeriLogger Pro/ WaveFormer Pro/Timing Diagrammer Pro** y tendrá precedencia cuando ocurra cualquier conflicto relativo a la licencia.

1. Este software fue desarrollado con fondos privados (ninguna parte de él se desarrolló con fondos gubernamentales) y es un secreto comercial de SynaptiCAD de acuerdo con los propósitos del Acta de Libertad de Información; es materia de “software de cómputo comercial” la utilización limitada prevista en cualquier contrato entre el vendedor y la entidad de gobierno; es propiedad exclusiva de SynaptiCAD.

2. Para unidades de DoD, el software se vende sólo con “Derechos restringidos” en tanto que ese término está definido en el suplemento DoD DFAR 252.227-7013(b)(3)(ii), y su uso, duplicación o divulgación es materia de restricciones especificadas en el subpárrafo (c)(1)(ii) de los Derechos sobre Datos Técnicos y Software Computacional, cláusula 252.227-7013. Fabricante: SynaptiCAD, apartado postal 10608, Blacksburg, Va 24062-0608 EU.

3. Si el software fue adquirido bajo el GSA Schedule, el gobierno está de acuerdo en abstenerse de cambiar o remover cualquier insignia o letra del software o su documentación, así como de producir copias de manuales o discos (excepto para propósitos de respaldo) y: 1) el título propiedad del software y la documentación y cualquier reproducción de ellos permanecerá en manos de SynaptiCAD; 2) el empleo del software se limitará al uso para el que fue adquirido; y 3) si el uso del software es discontinuado en la locación de la instalación original y el gobierno desea usarlo en alguna otra locación, lo podrá hacer avisando antes por escrito a SynaptiCAD, especificando la nueva locación y tipo de computadora.

4. El personal gubernamental que haga uso del software, que no sea aquél bajo contrato DoD o GSA Schedule, está enterado por este medio de que el uso del software es materia de restricciones que son las mismas o similares a las especificadas más arriba.